Interactive Process Models

Håvard D. Jørgensen
hdj@sintef.no

Department of Computer and Information Science

Faculty of Information Technology,
Mathematics and Electrical Engineering

Norwegian University of Science and Technology
Trondheim, Norway

January 7, 2004

Til Anniken

Abstract

Contemporary business process systems are built to automate routine procedures. Automation demands well-understood domains, repetitive processes, clear organisational roles, an established terminology, and predefined plans. *Knowledge work* is not like that. Plans for knowledge intensive processes are elaborated and reinterpreted as the work progresses. *Interactive process models* are created and updated by the project participants to reflect evolving plans. The execution of such models is controlled by users and only partially automated. An interactive process system should

- Enable modelling by end users,
- Integrate support for ad-hoc and routine work,
- Dynamically customise functionality and interfaces, and
- Integrate learning and knowledge management in everyday work.

This thesis reports on an engineering project, where an interactive process environment called WORKWARE was developed. WORKWARE combines workflow and groupware. Following an incremental development method, multiple versions of systems have been designed, implemented and used. In each iteration, usage experience, validation data, and the organisational science literature generated requirements for the next version.

Design ideas for WORKWARE are the main contributions of this thesis. *Semantic holism*, where the meaning of each model element depends on the rest of the model, makes the language simpler, more flexible and user-oriented. Processes are modelled at the instance level, to capture the unique structure of each project. Local models are harvested into templates, which can be reused in similar projects. Multi-dimensional classification structures are built incrementally, helping users to identify suitable models for their work. Modelling, harvesting and reuse thus form an organisational learning cycle, anchored in practice.

Interactive process models will be evolving and incomplete. We define *interactive enactment semantics* that automates well-defined parts of the model, and asks users to handle ambiguous parts. Users may also override the default interpretation. A language that explicitly models the *decisions* that control the flow of work, can be used for both structured and ad-hoc processes. Structure may be added or removed throughout the process, enabling users to plan their work with the level of detail that they find useful.

The WORKWARE prototype has been used by a number of projects, and integrated with other tools for modelling, simulation and real-time collaboration. Data from interviews and questionnaires show that the system is useful. Models developed by users illustrate its practical application. Comparison with existing systems shows that the interaction approach is novel, and that WORKWARE is simpler and more flexible than existing tools. The analysis also demonstrates that the designs are useful for interactive models of other domains as well. In addition to extending the concept of workflow to include emergent processes, this work thus has resulted in general techniques for interactive models that change while they are executed.

Preface

This thesis is submitted to the Norwegian University of Science and Technology for the degree of "doktor ingeniør". The work has been carried out under the supervision of Professor Arne Sølvberg at the Information Systems Group, Department of Computer and Information Sciences, during 1999-2003. I have also drawn upon preceding and concurrent research projects at SINTEF, most notably AIS (*Advanced Intranet Cooperation*, sponsored by the Norwegian Research Council NFR, 1997-1999) and EXTERNAL (*Extended Enterprise Resources, Network Architectures and Learning*, EU IST, 1999-2002). My studies were funded by the NFR project *Living Knowledge*.

The thesis would not have materialised had it not been for a number of talented colleagues, customers and supervisors. I wish to thank Arne Sølvberg for providing efficient advice, especially concerning research methodologies and approach. He has pointed me in the right directions without excessively detailed follow-up. Steinar Carlsen served as co-supervisor during the first year. He was instrumental in turning my interest to workflow, process modelling, knowledge management and CSCW. The seeds of many ideas presented here were sown during our collaboration around the WORKWARE prototype in the AIS project. In addition to his modelling language APM, Steinar already had ideas for user involvement in workflow enactment, and he came up with the term 'emergent workflow' for our first joint publication in 1998. When Steinar left SINTEF, John Krogstie replaced him as my co-supervisor. John brought unique expertise in conceptual modelling, and through our discussions my work also turned towards modelling in general, and the human, social and organisational issues intertwined in the modelling process. John's comments and careful reading also helped me structure and shape this thesis.

Colleagues at SINTEF have contributed to the WORKWARE system. Rolf Kenneth Rolfsen originally proposed the integration of awareness and workflow, and implemented parts of that component. Oddrun P. Ohren, Ole A. Brevik and Marit K. Natvig were involved in the design of the document manager, while Svein G. Johnsen was responsible for the interface to EXTERNAL's modelling language and repository. The EXTERNAL infrastructure further required the efforts and imagination of Dag Karlsen, Stefano Tinella, Jessica Rubart, Weigang Wang, Frode Thue Lie and others. Jörg Haake, Kristin Strømseng, Kirsten Krokeide, Heidi Brovold, Manos Georgoudakis, Manolis Chrysostalis, Kostas Giotopoulos, Guido Scagno, Jarle Hildrum and many of the aforementioned applied and evaluated our technologies in use cases. I am also grateful to Frank Lillehagen, whose notion of active knowledge models guided this work. Just as important was perhaps his continuous regeneration of creative chaos in our projects.

Table of Contents

Chapter 1
Introduction

The original objective of this research was to design a flexible information system for planning, coordinating, managing, and performing knowledge intensive work. The system was to be based on visual models of work processes. We also wanted to utilise these models better for organisational learning and knowledge management.

Requirements were gathered in workshops with prospective users, and from the literature on information systems (IS), computer-supported cooperative work (CSCW), organisational, social and human sciences. Case studies showed that although many view process support systems as useful tools, most users complain about the rigidity of software-controlled processes. Exceptions to modelled rules occur frequently, even in seemingly routine processes in offices and factories [527, 547]. In knowledge intensive work like engineering and consulting, changes to plans are expected throughout the projects.

Faced with such evolving and incompletely understood work, most process support systems are still based on the *closed world* assumption, where model execution is completely controlled by software. Models thus have to be formal, consistent and complete. When exceptions or model evolution violate the closed world assumption, complex algorithms are triggered in order to restore a consistent state.

We decided to take the opposite approach, to design an *open* system. Seeing that people dealt sophisticatedly with exceptions and change in everyday work, we aimed at augmenting these capabilities with tool support. Closed systems need more detailed and precise process models than most users are willing or capable to provide. If users can resolve model ambiguities at the time of execution, they need not model so much detail in advance. We called this approach *"interactive models"*, because it follows the notion of interaction as an extension of algorithmic computing [523].

In this approach, process modelling is moved from the domain of process experts and out into the work environment, to managers and work performers. Knowledge management and process improvement is given a foundation in models of individual cases, which better reflect the way work is actually performed. The role of process experts is to define templates that users can apply when needed in local models. In order to demonstrate that this approach is feasible and useful, three engineering challenges must be met:

1. *How can an IS support end users in modelling their own work?*
2. *How can an IS utilise evolving models to provide contextual work support?*
3. *How can models be reused, adapted and combined in new settings?*

We refer to these interdependent problems as model *articulation*, *activation* and *reuse*.

1.1 Problem: Knowledge Intensive Cooperation

This introduction seeks to justify two assumptions upon which this thesis is based:

1. Knowledge workers require more flexible IS than what is currently available.
2. Process models can be applied at runtime to achieve increased flexibility.

The term *knowledge intensive work* is a result of classification by dominant resource, distinguishing it from labour and capital intensive. Knowledge intensive work requires innovation by the workers to be successful [159]. Examples include consulting, engineering, and other forms of professional work [445]. Knowledge is an intangible, evolving resource possessed by individuals. It is highly context-dependent. We use and develop our knowledge in interplay with other people and objects in our environment. The knowledge of each individual is a unique result of his or her experiences. Differences in individual work experience, education, personality and social background enable teams to attack problems from a multitude of viewpoints. The uniqueness of individual knowledge is thus both the raison d'être and a main barrier to cooperation.

This duality is evident in information systems development. Managers, user representatives, business consultants, system analysts, designers, programmers etc. bring unique knowledge to the joint project. All this knowledge is vital to reaching the objectives. Communication, cooperation and problem solving across disciplinary boundaries require mutual learning and openness to the perspectives of others. Users must develop an understanding of technical opportunities and limitations, while developers must grasp the local reality of the work environment. Through open dialogue and joint experience, the project group constructs shared understanding and frames of reference. Conflicting goals, people working part time on the project, in different locations and organisations, disrupt the process of establishing an effective team. Rigidity and lack of interoperability among software tools constitute yet another barrier.

1.1.1 Flexible Support for Projects and Virtual Enterprises

Knowledge intensive work is frequently organised in projects. A project is a temporary organisation established to handle a unique problem [400]. Projects exist outside of conventional organisational units, bringing together people from different functions. In a recent survey of 1000 companies in Norway, 69% reported that they commonly organise work in projects [248]. In knowledge intensive industries like IT, consulting, oil and gas, around 90% of the companies are involved in projects. Another analysis concludes that work is becoming more knowledge intensive [88]. Global competition, increased and more diverse information needs, shortcomings of skill specialisation and larger knowledge components in products and services, are among the main reasons for this trend. Most future economic growth is expected to occur in the service sector.

Inter-organisational projects are as common as internal projects [248]. Small and medium sized companies, in particular, are frequently involved in inter-organisational projects. The term 'virtual enterprise' (VE) often designate such arrangements. According to a recent white paper, VE process management tools are currently "*obtuse and inaccessible to the vast majority of knowledge workers*" [126]. The Internet and other information systems increasingly enable outsourcing, e-business, and globalisation. E-Commerce frameworks [450], Internet portals, workflow management systems [12, 408, 553], groupware [404], supply chain management, and enterprise resource planning systems (ERP) [294] all contribute to this trend. While these systems automate routine transactions, knowledge-based cooperation remains a challenge. Flexibility is thus an important research topic in all of these areas, and alongside alignment with local procedures it is the most important criteria for IS selection [160].

Paradoxically, studies conclude, "*simple and adaptable technologies enable more complex virtual collaboration*" [404]. Low level tools like email, are used far more frequently than sophisticated coordination systems. A survey reports that 82% of the companies "very often" use email for such purposes, while only about a third use a project management system [248]. It also uncovers that knowledge intensive industries make far more use of information technology than the less knowledge driven. This suggests readiness as well as need for new technologies.

1.1.2 Integrating Support for Routine and Ad-Hoc Work

Figure 1 [192] shows the scope of current coordination systems with respect to different organisational forms and relationships between collaborating partners. Where there is high uncertainty about the actual content of the work (as in typical knowledge intensive processes), projects are the dominating organisational form, and groupware the primary coordination technology. In more stable and rule-based hierarchies, workflow management systems dominate. Electronic commerce deals with the exchange of well-defined goods and services, where there is typically low goal congruence among the cooperating partners. Consequently, requirements for security and transaction support are strong.

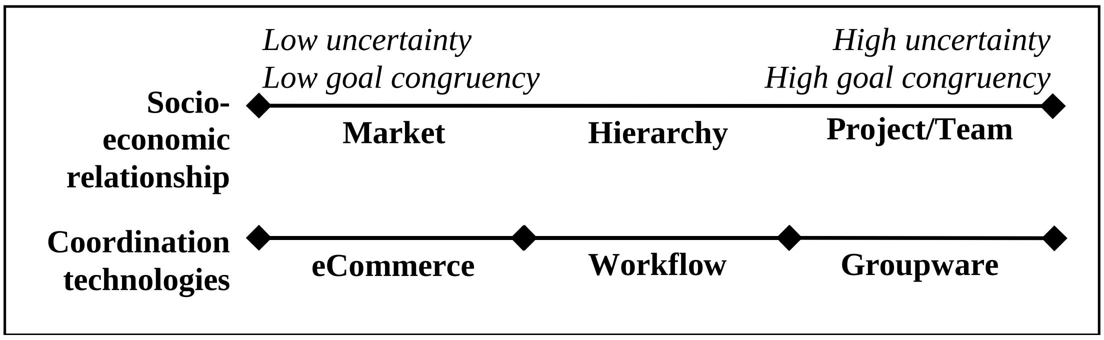


Figure 1. Coordination technologies for different forms of collaboration [192].

Few work processes reside completely in one of these categories, however. Many have aspects of inter-organisational collaboration in that goods or services are exchanged; most have routine aspects in administration, reporting and accounting; and all knowledge intensive processes do by definition have creative, non-routine aspects. While process support systems have proven useful in automating routine procedures, support for the increasingly important knowledge intensive processes, the remaining 80% [126], remains a challenge. A workflow industry benchmarking initiative thus concludes that integrated coordination support across process types is the most important challenge [390]. Realising that one size does not fit all, dynamic *process diversity* [314] is advocated along this *planning spectrum* [56]. Users must be supported in selecting a suitable degree of plan specificity for the current state of their project, balancing plan complexity with the need for coordination, guidance, and control.

1.1.3 Integrating Knowledge Management in Everyday Work

Knowledge management (KM) is the collection of processes that govern the creation, dissemination and utilisation of knowledge to fulfil organisational objectives. The term is popular, but often misused [106]. This may in part be caused by the apparent contradiction in the term. After all, how can an intangible resource be managed? Many companies recognise the value of knowledge only once it is gone, e.g. as a result of downsizing or process automation [122]. Although failure rates of above 50% have been reported, 80% of large corporations have KM projects [301].

Knowledge is defined as *"justified true belief"*, or more precisely as *"a relatively stable and sufficiently consistent set of conceptions, possessed by single human actors"* [162]. A more dynamic view defines *knowing* as *"an active process that is mediated, situated, provisional, pragmatic and contested"* [53]. Knowledge is not represented in artefacts, though the data stored in computer systems have a potential for creating knowledge, if a continuing dialogue between people and artefacts is nurtured, if the representations are used actively in work, communication, and learning [527].

Knowledge management tools "*enhance and enable knowledge generation, codification, and transfer*" [419]. Many approaches over-emphasise codification, seeking to document experience [424]. Post-mortem analysis [52] by definition comes too late to help the project. Such approaches emphasise the most manageable aspects of knowledge, and separate knowledge management activities and roles from the core work. Increased focus on the social processes of learning, on knowledge management intertwined in work practice, is required [122, 441]. After all, knowledge is perhaps the only resource that increases when used. Need-driven, just-in-time knowledge transfer has thus been advocated [279]. In the area of process knowledge, such approaches currently lack tool support.

1.2 Approach: Interactive Process Models

Models are defined as explicit representations of some portions of reality as perceived by some actor [524]. A model is *active* if it influences the reality it reflects; if changes to the representation also change the way some actors perceive reality. *Model activation* is the process by which a model affects reality. Activation involves actors interpreting the model and adjusting their behaviour to it. This process can be

- *Automated*, where a software component executes the model,
- *Manual*, where the model guides the actions of human actors, or
- *Interactive*, where prescribed aspects of the model are automatically interpreted and ambiguous parts are left to the users to resolve, with tool support.

Fully automated activation implies that the model must be formal and complete, while manual and interactive activation also can handle informal and evolving models. We define a model to be *interactive* if it is interactively activated. The process of defining and updating an interactive model is called *articulation*. In this thesis we are primarily concerned with interactive models of work processes. By altering models of their own work, users can control and customise the behaviour of an interactive system. The interplay of articulation and activation (Figure 2) keeps the models alive and up to date as resources for learning, coordination and work support. The three engineering challenges (articulation, activation, and reuse) all contribute to increasing the benefits of interactive models and decreasing the efforts and learning required to use the system.

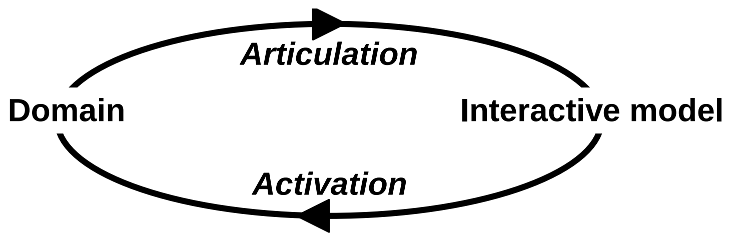


Figure 2. The interplay of articulation and activation.

1.2.1 The Potential of Interactive Process Models

The constantly changing nature of the competitive environment in the global network economy [266] creates *emergent organisations*, where "*every feature of social organisation - culture, meaning, social relationships, decision processes and so on - are continually emergent, following no predefined pattern*" [497]. This environment requires evolving information systems, adapting their behaviour to updated models of the usage environment.

Articulation: Simple and User-Oriented Process Modelling

Our approach relies on the assumption that end users must be actively involved in creating, updating and interpreting models of their own work, as part of the work. Local participants are the only ones with sufficient knowledge of the process. Modelling by end users has met scepticism from the workflow research community [46]. On the other hand, studies of user participation in IS development, tailoring, knowledge management and process improvement indicate that our approach is viable [19, 468, 527]. In workflow management, users also deal creatively with change and exceptions, often by taking the process out of the system and handling it manually [69]. Systems not designed for user involvement thus present a barrier to local innovation, and are unable to capture these contributions for further assessment and knowledge management. End user participation remains primarily an organisational problem, involving trust, power and community building, but simple, user-oriented, and adaptable modelling languages will remove many barriers.

Activation: Customised and Integrated Software Support

Simple and useful tools motivate use. Information systems that offer a wide range of functionality often become overwhelmingly complex and incomprehensible. Consequently, only a small portion of the available functionality is utilised. This condition is known as *featuritis*. We need role and task specific user interfaces, containing just what is needed in the current context. Interfaces and semantics should also adapt to the local needs of each project. Process models, articulating who performs which tasks when and why, is a powerful resource for such customisation. Systems and processes should also adapt to the skills and preferences of each individual. Personalisation fosters a sense of ownership, further motivating active participation.

In virtual enterprises, the unique nature of each project, and the changing set of partners, seldom makes it economically viable to integrate information systems through conventional development methods. Standardisation requires that the domain is static and well understood, and is thus seldom appropriate for knowledge work. Consequently, we need a flexible infrastructure that allows shared understanding and semantic interoperability to emerge from the project, rather than being a prerequisite for cooperation. Interactive models provide a simple, visual approach to capture shared understanding as it unfolds.

Reuse: Process Knowledge Management

The gap between what people say and what they do, makes it difficult to use plans and other official descriptions of work as input to KM [19]. Local articulation of process models must thus be straightforward, but still some knowledge cannot be modelled and will remain tacit [370]. Process models will thus be incomplete while they are used, subject to an ongoing elaboration and interpretation. Models are completed only when they are no longer in use. Interactive modelling allows the system to handle incomplete, evolving descriptions of work, by involving users in resolving incompleteness and inconsistencies during activation. The openness of the approach allows local process innovation to be captured, assessed and packaged for reuse in similar future projects.

1.2.2 Research Objectives

Interactive process models combine model-driven groupware and workflow support (cf. Figure 1 on page 3), in order to support knowledge intensive work. Model-driven functionality is already well established for handling routine procedures, so the remaining challenge is to apply models to customise support for projects and virtual enterprises. In addition to the engineering challenges outlined above, this work also seeks to

Extend the concept of "workflow" to include ad-hoc, unstructured processes.

Most research in conceptual modelling has focused on system development and business engineering. Some studies explore *models in use*, immersed in an organisational setting, but seldom apply the insights to design new systems. We thus also aim to

Develop generic design principles for models that evolve while they are executed.

These objectives guide the work towards more lasting contributions, turning what could have been just a development project into research.

1.3 Research Method

This is an engineering thesis. The main objective of IS engineering is to improve the methods and practice of IS development. Empirically oriented scientists currently challenge the position of engineering research in this area [463, 490, 545]. Appendix A provides an in-depth discussion on the methodological basis for this thesis, arguing that there is a need for interpretive engineering research, that measurable, repeatable experiments with well-defined hypotheses are insufficient for solving current IS problems.

Development of an information system is a *wicked problem* [482] without a generally agreed upon formulation, which cannot be stated as well-defined hypotheses. Wicked problems are unique and novel; require complex judgement; have no objective measure of success or completion, no right or wrong solutions, and no given alternatives [455]. To attack wicked problems, *problem setting* (understanding) and *problem solving* must be intertwined in a process described as *reflective practice* [445]. The *repertoires of action* that engineers have learned through practice and education are crucial both for problem setting and problem solving [445]. These repertoires contain exemplary solutions, design patterns, approaches and perspectives. Reflective engineering research creates new knowledge not by giving rise to general principles, but by adding to the practitioners' repertoire.

1.3.1 Research Approach

Figure 3 illustrates the research method of this thesis. It is based on an incremental development cycle, where *requirements* guide *design* and *implementation*, and prototype *use* generates new requirements. The general approach of interactive models

- helps us to frame the problem,
- acts as a metaphor that generates design ideas,
- integrates the components of the implementation, and
- enables users to customise the system during operation.

The approach is thus validated in all phases of system development. This engineering cycle integrates validation, in that the implementation shows that the designs are feasible, usage shows that the system fulfils needs etc. Experience from all phases is continuously utilised to improve the next release of the system.

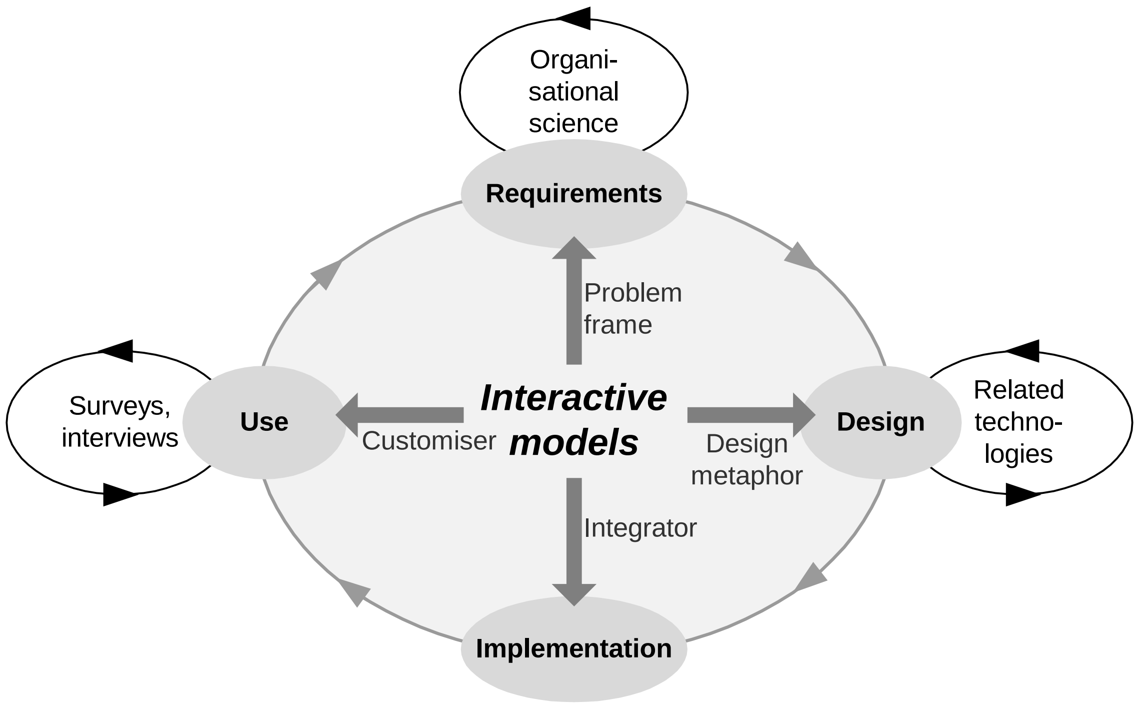


Figure 3. Overview of the research approach.

The approach follows principles of reflective practice (Appendix A.2.1 [445]). In addition to the core cycle's *reflection in action*, a number of validation activities turn the development project into research through *reflection on action*. The generality of the requirements is validated by a theoretical grounding in organisational and social sciences, while comparisons with previous technological research justify that the designs are innovative and relevant. Surveys and interviews indicate that the approach, design and implementation do meet user needs and expectations.

1.3.2 Research Context

To enable a rich understanding of a research project, its environment and background should be described. This thesis continues previous work by Steinar Carlsen on flexible workflow modelling [90], where organisational requirements were gathered [92] and existing systems evaluated [96]. This led to the definition of the APM modelling language [90, 91]. My work started as an attempt to build a runtime system for APM. Three research projects have served as main arenas for user participation, conceptualisation, design, and experimentation:

- *AIS* (Advanced Intranet Cooperation, 1997-1999) explored the use of Internet technologies for information sharing and cooperation support. It involved two large user organisations, an oil company (Saga) and a maritime classification, certification and consulting company (DNV), an enterprise modelling tool vendor (Metis), a technological (SINTEF) and a social science (FAFO) research institute. I here developed the first WORKWARE prototypes to help the customers to assess the usefulness of Intranet process support systems. Users contributed requirements in a number of workshops. Ideas and strategies from Metis, in particular the AKM (Active Knowledge Model) metaphor [310], also guided our work.
- *Living Knowledge* (1999-2002) was a multi-disciplinary project investigating organisational learning. The common perspective of this project emphasised knowledge in use, within a particular context. Living knowledge has cultural roots; it is tacit, distributed and socialised. It is about how and why, more than what, and is not primarily represented in computer systems. From this perspective, we investigated knowledge management strategies and tools as well as the nature of knowledge work.
- *EXTERNAL* (Extended Enterprise Resources, Networks and Learning, 2000-2002) [161] was a European project. It aimed to support the whole lifecycle of a virtual enterprise, from inception, planning, working, management and coordination, to decommission. In addition to the AIS partners DNV, Metis and SINTEF, the project involved another technological research institute (Fraunhofer IPSI, Germany), and a network of small and medium sized IT companies (Zeus EEIG, Greece). In EXTERNAL, interactive process models were applied in three cases: business consulting, software development, and research.

A wide range of perspectives from industry and research has thus influenced the ideas presented in this thesis. My previous education (information systems engineering) and work experience (developing software products) also probably oriented the work towards practical aspects of systems development.

1.3.3 Cooperation in Interdisciplinary Projects

Research and system development are cooperative activities. By involving more people in requirements collection, design, implementation and evaluation, the likelihood of successful products and balanced validation increases. While cooperation brings strength to validation, it makes individual's contributions harder to discern[1]. This thesis draws heavily on joint work in validation, while its contribution centres on design ideas and approaches proposed and systematised by the author. The EXTERNAL cases were studied by sociologists [37], an anthropologist [426] and a political scientist [287, 312].

[1] Acknowledgements of other people's contributions to this work are found in the thesis preface. In the rest of the text, joint work and contributions from others is explicitly stated and/or referenced.

This thesis is thus part of a larger body of interdisciplinary research anchored in real world settings. My role in these studies was to provide customisation and technical support, not to model, perform or evaluate the work. In addition to data gathered by others, the discussion in this thesis reflects my interpretation of the findings, and my exploration of the process models and customisation requests articulated by the users.

1.3.4 Validation

"*Every human tool relies upon, and reifies, some underlying conception of the activity that it is designed to support. As a consequence one way to view the artefact is as a test of the limits of the underlying conception*" [468]. Different contributions call for different validation techniques. For contributions to modelling languages and concepts, prototype implementation and subsequent experimentation is adequate. However, as the next section shows, the contributions of this thesis have a wide scope. A thorough data collection on all relevant attributes of each technique, is a too overwhelming task for one thesis. Hence, the central characteristics must be identified. This is the topic of the remainder of this section.

Comparative Evaluation

Related work helps to identify innovative aspects of different contributions, and to verify that they indeed are novel. It also shows the relevance of these innovations to already studied problems. These criteria are applied:

- *Simplicity*, measured by the number of constructs in the modelling language, the number of metametamodel concepts and operations, and the number of elements in typical models.
- *Flexibility*, which changes the system can handle, the number of operations involved in typical evolution patterns, and the range of exceptional scenarios that are tolerated without changing the model.

Sociotechnical systems [357, 495] is a relevant overarching theory for information systems. IS are sociotechnical, and process support systems influence job design, so sociotechnical job design principles are suitable for assessment of these technologies.

Gathering Usage Experience

The use cases of the EXTERNAL project [161] validate many features of the WORKWARE prototype and the interactive modelling approach. Interviews and questionnaires complement the more detailed comparative evaluation described above. The cases were selected after many of the contributions were completed, so there is little risk of self-fulfilling specialisation of the solution, e.g. that it is made specifically for the case, and only performs well on this case. Differences among the cases further increase confidence in this validation.

Empirical studies of reusability and adaptability are inherently difficult and time-consuming. There are generic problems of field studies (monitoring multiple persons in different locations over time), but also additional complexities stemming from the fact that the context and content of the work is expected to change when organisations apply our tools. Such long-lasting field studies are doctoral projects in their own right. Positivist quantitative and controlled experiments, while offering scientific rigour, cannot accurately reflect prolonged social processes of learning and cooperation. The complexity, uncertainty and dynamics of this domain require interpretive methods. It is a more realistic objective to capture process models from the cases, and to look for occasions of reuse and adaptation in their change histories.

1.3.5 Scope

Clearly, any solution includes organisational, social and human aspects of the usage context. While engineering research must be based on a thorough understanding of the usage environment, it provides new theories and experience primarily related to the *design* of computerised information systems. The key objective is to provide integrated support for interactive process model articulation, activation and reuse. A complete solution is beyond the scope of a single thesis. Instead, this work focuses on the core aspects of interactive process models, the modelling language (for articulation), semantics (activation) and underlying framework (reuse).

Although the work is focused in this direction, the wider picture still holds important requirements for a complete solution, and our design has numerous extensions in mind. For instance, the interactive architecture can integrate additional model-driven software components, and the modelling framework limits the complexity of multi-purpose models. Every aspect of the solution is designed with the premise that the developer only has an incomplete view of what users will need. Consequently, no component assumes to be in complete control, and customisation and extension mechanisms are essential. The system is designed to be open.

1.4 Contribution

The contributions of this thesis fall into different categories [166], including a general approach, new explanations and perspectives, new functionality and new implementation techniques. They bring us closer to meeting the three engineering challenges and the two research objectives.

1.4.1 Articulation

The feasibility of end users modelling their own work is contested. Some researchers claim that end users cannot be trusted to change workflow models correctly. We survey case studies where such innovations did occur, and our usage experience further adds to the evidence. The WORKWARE process modelling language further illustrates that a simple, flexible and user-oriented modelling languages can be based on *semantic holism*. Holism implies that the meaning of each model element depends on the rest of the model. We identify holistic modelling techniques like properties, constellations and context-sensitive semantics.

1.4.2 Activation

Customisable, contextual support for ad-hoc and routine processes, is enabled by

- *Interactive activation*, involving users in the interpretation and execution of incomplete and evolving models. This is achieved by representing, as model elements, the decisions involved in interpretation.
- *Holistic activation*, where the interpretation of each model element depends on the current states of surrounding elements, enabling a richer, more situated execution.
- *Visual, domain-oriented tailoring*. In an interactive model, domain and user oriented concepts are applied for tailoring the system, not just system oriented terminology.

1.4.3 Reuse

A need-driven approach to process knowledge management is anchored in models developed locally in each project. Local models are harvested and packaged for reuse. This is enabled by a modelling framework that supports

- *Incremental classification with instances as primary objects*, where instances locally define their own structure and behaviour. Classification structures are not part of the object definition, and can thus be built dynamically to reflect multiple perspectives.
- *Generalised inheritance*. Guided by requirements, we allow inheritance along any relationship in the model, not just specialisation.
- *Reuse policies*, which control the propagation of *reusable aspects* along modelled relationships, allowing users to customise the inheritance mechanism.

1.4.4 Workflow Concept

Where contemporary workflow management systems provide automatic activation of process models, the interactive approach involves users in interpreting what the model means in the current context. Bottom up articulation complements top down control, integrating ad-hoc and routine processes.

- The difference between adaptive and *interactive, emergent workflow* is described, clearing some of the confusion that has prevailed in this area [46, 274, 276].
- An *interactive, multi-aspect workflow architecture* integrates multiple model-driven software components, each offering complementary interpretations of the model. The components are called *interactors*, because they integrate articulation and activation, rather than separating them, as in contemporary architectures. The range of model driven functionality is thereby extended beyond workflow enactment.

1.4.5 Interactive Modelling Design Principles

Interaction was introduced by Wegner [523] as a theoretical framework for understanding computing. Here interactive models are proposed as a *design metaphor* for flexible information systems. We elucidate the concept of interactive models, uncover requirements, and identify suitable modelling techniques, e.g. semantic holism, instance modelling, and explicit representation of decisions.

1.5 The Structure of this Thesis

This thesis consists of ten chapters. The structure linearises the engineering research method cycle (Figure 3). This introduction has outlined the central objectives, problem definition, approach and contributions. Chapters 2 and 3 provide the background for the work, while 4, 5, and 6 contain the contributions of this thesis. The validation of the proposed designs is found in Chapters 7, 8 and 9.

Chapter 2 - Problem Setting elucidates basic concepts and differences between interactive models and models for systems development. We then explore the social context in which models are used. This survey leads to a list of requirements.

Chapter 3 - State of the Art. Here existing workflow and modelling approaches are analysed based on the list of requirements. This points to a number of limitations, especially with respect to simplicity, flexibility, and comprehensibility.

Chapter 4 - Interactive and Emergent Workflow defines a simple and user-oriented modelling language, as well as its interactive activation semantics. We also present an open, interactive architecture that combines workflow and groupware services.

Chapter 5 - Reuse specifies the modelling constructs and mechanisms that enable model reuse. An instance-based metametamodel allows local modifications, while user defined policies control inheritance along modelled relationships.

Chapter 6 - Interactive Modelling Techniques generalises the designs of the earlier chapters, identifying techniques that are relevant for interactive modelling outside the particular area of process models.

Chapter 7 - Implementation describes how the WORKWARE prototype implements the interactive architecture and language formalisms. In the EXTERNAL project WORKWARE was integrated with modelling, simulation, and real-time collaboration tools.

Chapter 8 - Usage Experience. The infrastructure from Chapter 7 has been applied in virtual enterprises doing research, business consulting and software development. Experience from these studies demonstrates the usability of WORKWARE.

Chapter 9 - Related Work compares WORKWARE to existing modelling frameworks, activation schemes and reuse mechanisms. The analysis shows the originality and relevance of this work for different research areas.

Chapter 10 - Summary and Further Work gives an overview of the contributions and point to interesting directions for further research into interactive models, emergent workflow and process knowledge management.

Chapter 2
Problem Setting

Interactive models constitute the main problem frame of this thesis. Such models are available to the users at runtime. They control the behaviour of the system, and thus allow users to adapt it to local needs and changing environments. Interactive models are immersed in day-to-day work. The development of such technologies requires in-depth understanding of social organisations, work, cooperation and learning. This chapter surveys organisational and social science literature in order to understand business and user needs and derive system requirements that address these needs. We first establish the basic terminology. The three engineering challenges, model articulation, activation and reuse, are then discussed in turn. This discussion aims to justify that the following propositions make sense:

1. Interactive models constitute a suitable problem frame for flexible IS (section 2.1).
2. Knowledge workers are capable of articulating, coordinating and reflecting on their own work. Those who perform the work should be involved in modelling it (2.2).
3. Planning and performance of knowledge work is intertwined, plans evolve, and all work has both routine and ad-hoc parts (section 2.3).
4. Evolving, incomplete and semi-formal models are needed in order to mediate workers' knowledge (sections 2.3-2.5).
5. Knowledge management needs closer integration with work practice (section 2.4).
6. Social learning theories help us understand the process of modelling (section 2.4).
7. Motivating end user participation in modelling is a core challenge (section 2.4.4).
8. Processes are central to work and organisation (section 2.5).
9. Process knowledge management is not primarily about learning to follow rules; it is about interpreting guidelines creatively in the situations that arise (section 2.5.7).

2.1 Basic Concepts

In section 1.2, the basic concepts of this thesis were introduced. A *model* is *active* if it influences the reality it reflects. Model *activation* is *automatic*, *manual* or *interactive*. That a model is *interactive* (interactively activated) entails a co-evolution of the model and the domain. A model that does not change cannot reflect aspects of reality that change, nor can it reflect evolution of an actor's understanding. Consequently, an interactive model that does not evolve will deteriorate. Fully automated activation in a closed system can in theory avoid this problem, but cooperative IS are open systems. Interactive activation involves autonomous human actors, so the models will need to evolve. The process of updating an interactive model is called *articulation* [438]. The interplay of articulation and activation reflects the mutual constitution of interactive models and the social reality they reflect (Figure 2 on page 5).

Reuse refers to applying a model or model fragment in a situation other than the one it was originally defined for. *Templates* are model fragments especially adapted for reuse. *Harvesting* refers to using the models and experience from one or more local processes to generate a new template, a variant or a revision of an existing template.

2.1.1 Interactive Models in Workflow Management Systems

Let us illustrate these concepts with an example. Workflow management systems [532] support coordination of work based on explicit process models. Such models reflect the tasks that are part of the process, their interdependencies and the resources that are applied to perform them. Resources include personnel, information and tools. All workflow systems include an enactment service that *activates* the model. Most systems also include a process definition tool that lets users *articulate* models. In *static* workflow, process models are built by experts and not allowed to change while process instances are being enacted. Supporting fully automatic activation, static workflow applies to well-understood, stable, routine processes. *Adaptive* (or dynamic) workflow is an important research area [90, 276, 553]. Here the models are allowed to change, and change will affect ongoing processes. In *emergent* workflow (Chapter 4), modelling is viewed as an integral part of the work, performed by the process participants. The domain consists of unique projects. Emergent workflow views enactment as an interactive process, involving users in the situated interpretation of workflow models. This shift from automation to interactive activation enables the system to support incomplete and evolving models, better matching the contingencies of work.

2.1.2 Background

Conceptual modelling research has concentrated on system *development* activities. Models of work environments are applied to analyse the problem domain, to capture and structure user requirements. Most approaches focus on *referential* aspects, on the relationship between model elements and the real world objects that they represent. Individual, social and situational aspects of model *usage*, and *relational* theories of meaning [188], have had less influence upon mainstream information systems engineering [224].

There are notable exceptions. In 1986, Hewitt pointed out that offices are best described as open systems, where people cope with conflicting, inconsistent and partial information [218]. He showed the shortcomings of Turing based notions of computing in analysing such systems, and forecasted that future information systems would acquire more of the characteristics of human organisations, e.g. concurrency, decentralised control, indeterminacy, interconnectivity, and contextuality. Goguen [189, 190] point to the situated, local, emergent, contingent, vague, and open nature of model domains, and outline ramifications of this insight for requirements engineering. Like wicked problems attacked with reflective practice (cf. Appendix A.2.1), he concludes that requirements become clear only when a system is successfully operating [190]. Lillehagen [310] discusses the extension of enterprise modelling (articulation and manual activation) to *active knowledge models* (AKM) with automated support. Willumsen [535] develop execution support for conceptual models in information systems development, including users as actors contributing to the interpretation of the models.

More recently, Chen et al. [102] points to active models as a major direction for future research in conceptual modelling. Model execution, end user participation, and interaction are highlighted among the main challenges. Greenwood et al. [198] argue that active models enable systems to meet business needs that current technologies fail to solve. They present a notion of active models similar to the one given above, but with more emphasis on the active relationship between the domain and the model, and less on articulation. Their work is directed at devising methodologies for active software process support.

The most comprehensive approach to this field, however, is the interaction framework proposed by Wegner [522-525]. This framework is motivated by a number of current trends in computer science: The shift from standalone PCs to network computing, from procedural to object oriented programming, from closed to open systems etc. Its development was triggered by the realisation that machines involving users in their problem solving can solve a larger class of problems than algorithmic systems computing in isolation [523]. The primary characteristic that differentiates an *interaction machine* from a conventional Turing machine is that it can pose questions to users during its computation. As shown in Figure 4, computation is conventionally portrayed as a user providing input to the machine, which then processes the request and provides an answer (output). This stimulus-response model excludes interactions that could be used to establish shared meaning [190]. Moving from a Turing machine to an interaction machine, computation becomes a multi-step conversation between the user and the machine, each being able to take the initiative. Hence, research should not solely be concerned with the development of more powerful algorithms, we should also look at new ways in which the computerised and human parts of the system can cooperate in order to solve problems.

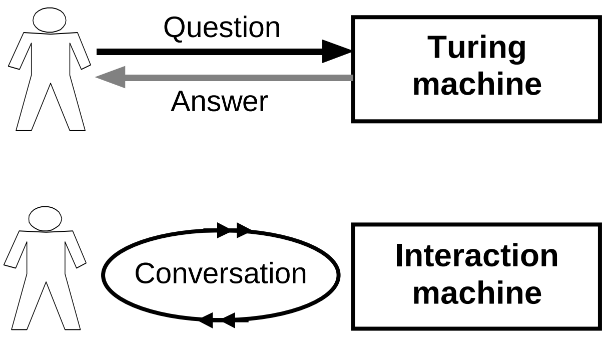


Figure 4. Turing machines and interaction machines.

Although researchers have pointed to aspects of interactive models, and put forward a theoretical framework for interactive computing, there is a lack of engineering research developing and validating interactive models as a design approach. Consequently, there has been little investigation into what are the new challenges of interactive models, and what modelling techniques are useful. The model quality framework of Krogstie, Sindre and Lindland [284, 291] allows us to illuminate differences between requirements for interactive models and requirements met by systems development models [289]. Such an understanding is important for assessing the application of conventional modelling techniques to interactive models. This framework is closely linked to linguistic and semiotic theory, and based on a social constructivist view (articulated below). The main concepts of the framework are shown in Figure 5, including the following sets of statements:

- L, the language extension, all statements that can be articulated in the language.
- D, the domain, all possible statements about the situation at hand.
- M, the externalised model, all statements in the explicit representation.
- K, the relevant explicit knowledge of the modeller.
- I, the social actor interpretation, what the audience perceives the model to say.
- T, the technical actor interpretation, how software components interpret the model.

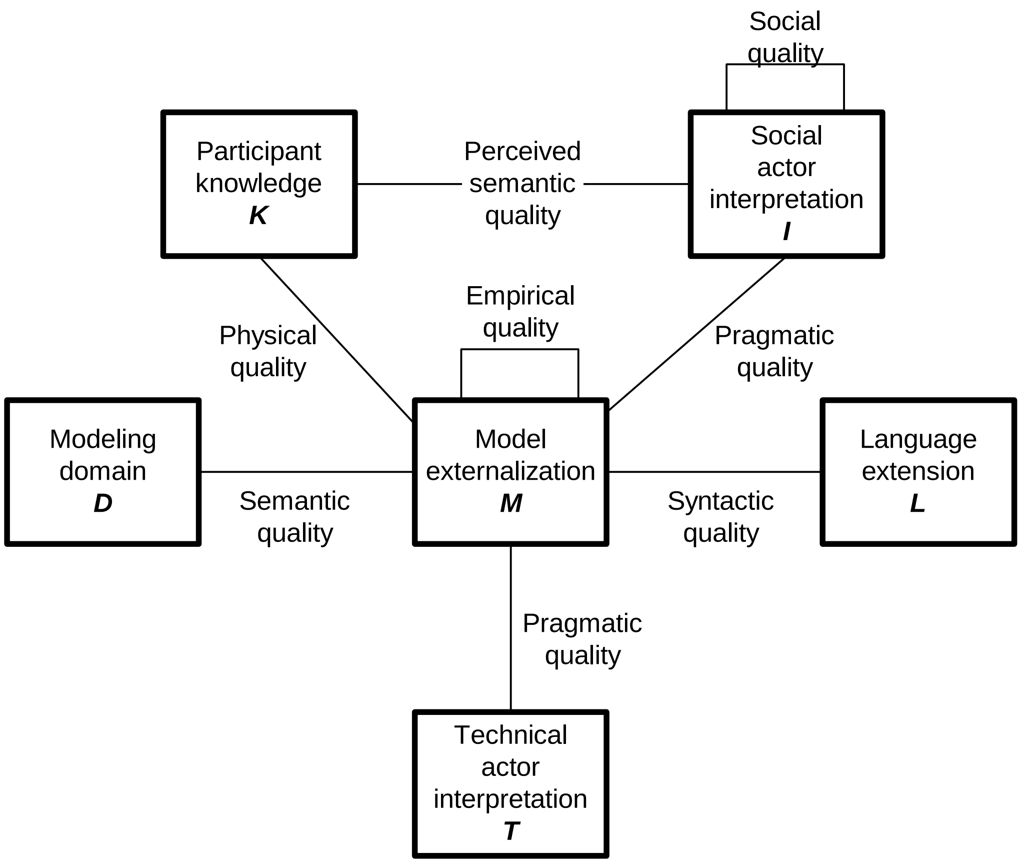


Figure 5. Framework for analysing the quality of models.

Model quality types are determined by the relationships between the statement sets:

- *Physical quality*, involving
  - Externalisability (articulation), that the relevant explicit knowledge of the participants is reflected in the model.
  - Internalisability, that the persons involved can make sense of the model.
- *Empirical quality* deals with error frequencies when a model is read or written by different users, coding, and ergonomics of human-computer interaction.
- *Syntactic quality* is the match between the model and the language.
- *Semantic quality* is the correspondence between the model and the domain. The framework contains two semantic goals:
  - Validity, that all statements in the model are correct and relevant to the problem,
  - Completeness, that the model contains all relevant statements about the domain.
- *Perceived semantic quality* is the match between the participants' interpretation of a model and his or her current explicit knowledge.

- *Pragmatic quality* is the correspondence between the model and the audience's interpretation of it.
- *Social quality*: The goal for social quality is agreement among different participants' interpretations. Social quality affects communication among participants about the contents of the model.

Since the domain D cannot be completely known, semantic quality can only be tested indirectly through the participants' knowledge.

Interactive and Passive Models

A conventional model used during the early phases of systems development is most often passive during system operation. Interactive models are faced with a different set of requirements than development models. A wider range of people are involved in the modelling, amplifying the social, psychological and organisational aspects. These differences have received little attention from active modelling research. Often solutions developed for systems development are simply transferred to active modelling. The use of UML (Unified Modelling Language) [229, 315, 335], or even programming languages [389, 541], for enterprise modelling and workflow, exemplifies this. Although the quality framework captures most of the perspectives relevant for interactive models, some adjustments are needed [256, 289]:

- An interactive model is immersed in its usage context. Model articulation and activation take place concurrently, and are mutually dependent upon each other. Agreement among actors is vital for IS development, because it is costly to fix errors late in the project. For an interactive model the costs of fixing errors is diminished. The goal of social quality for interactive models thus becomes to support social learning and construction of shared understanding.
- Similarly, semantic and syntactic correctness becomes less important because users can be put in charge of resolving inconsistencies during interactive activation. For learning, the ability to represent inconsistent points of view is crucial. A system should thus not deny articulation of syntactically incorrect model fragments, but instead capture inconsistent views so that they can be negotiated.
- Interactive activation implies that the goal of semantic completeness is replaced by a goal of letting the users articulate their reality at the level of detail and specificity that they find useful. Incompleteness can be resolved at the time of activation.

More fundamentally, the existing model quality framework embodies a static, structural perspective. It discusses relationships between sets of statements about the world. Interactive modelling quality [256, 289] demands a more dynamic approach, focusing on how these sets of statements interact and influence each other in the interdependent processes of articulation, activation and reuse. This perspective is conceptualised below, defining core processes as sequences of activities that transform statement sets in the quality framework model. A focus on core processes rather than static quality parameters emphasise interdependencies and trade-offs which designers of interactive systems must resolve. Figure 6 summarises this perspective. The rest of this section elucidates basic concepts in this framework. Below, capital letters refer to Figure 5.

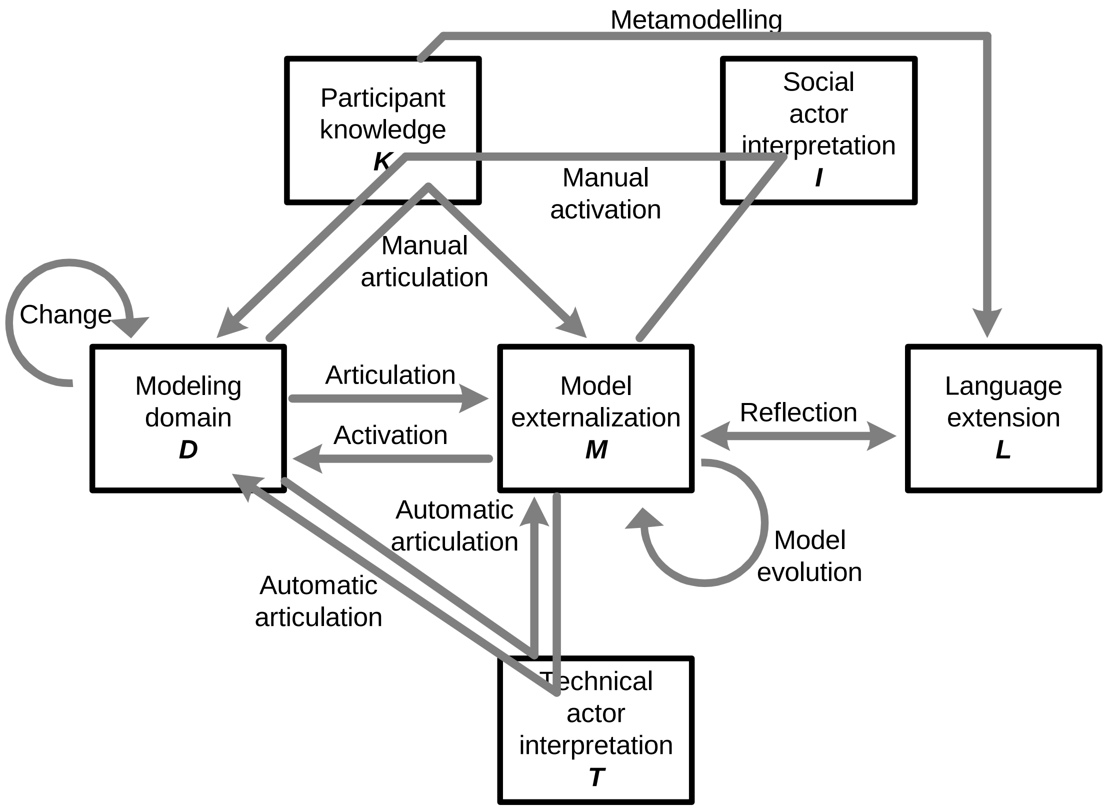


Figure 6. Interactive processes in the model quality framework.

Articulation (D→M)

Articulation is the process where domain features are represented in the model (M) by means of the modelling language (L). It should increase the semantic quality of the model.

Manual Articulation (D→K→M)

Most articulation is manual, involving the externalisation of participant knowledge (K) about the domain (D) into the model (M) using the language (L). In addition to empirical, physical and semantic quality, it depends on the quality of participant knowledge about the domain, and the appropriateness of the language with respect to the domain and participants' knowledge (comprehensibility and articulation appropriateness [90]).

Automatic Articulation (D→T→M)

Sensors are computerised components that capture information from the model domain, through interfaces to external information systems or hardware devices.

Change (D→D)

Changes in the domain (D) may become known to the participants of the project (K). Known changes may become articulated into the model M.

Model Evolution ($M \rightarrow M$)

Model evolution may be triggered by change and reflected through articulation, or it is motivated by the need to cause future domain change through activation.

Activation ($M \rightarrow D$)

Model activation involves model-guided actions that transform the domain (D).

Automatic Activation ($M \rightarrow T \rightarrow D$)

Automatic activation implies that the model is interpreted and acted upon by a computerised component, in our terminology a model *interactor*. The *executability appropriateness* of the modelling language [90] reflects its automatic activation potential.

Manual Activation ($M \rightarrow I \rightarrow D$)

Human actions based on an interpretation (I) of the model (M) constitute a manual activation process. The involved actors may also have created the model themselves (being participants (K) in the modelling). Actions involve changing the domain D, and should thus be reflected in the model M. While automatic activation is deducible from the model and thus already captured, manual actions include elements of human model interpretation, which should be captured in the model.

Interactive Activation ($M \rightarrow I \leftrightarrow T \rightarrow D$)

Interactive activation exists in different forms. In many workflow systems, it is the technical component that tells the users what to do (e.g. by putting tasks in their inboxes). In other words, the modelled sequence of tasks is automatically interpreted, but the tasks are performed manually ($M \rightarrow T \rightarrow I \rightarrow D$). In graphical user interfaces, the opposite sequence ($I \rightarrow T \rightarrow D$) is more common. Here it is up to the user to select which operation the system should perform next. If an action is the result of interactive or automatic model activation, its effects should be automatically captured in the model. This scheme increases the semantic quality without extra work for the users, completing the cycle $M \rightarrow I \leftrightarrow T \rightarrow D \rightarrow M$.

Reuse ($M_1 \rightarrow I_1 \rightarrow K_2 \rightarrow M_2$)

Reuse involves the application of a model (M_1 of D_1) in a setting different from the one where it was originally developed. Reuse thus involves an audience (I_1) who have to interpret and adapt a previously developed model to fit their local needs. The audience becomes participants (K_2) in the adaptation of the reused model (M_2) to the new domain (D_2). Social pragmatic quality is crucial for reuse, especially if the people involved did not participate in the articulation of the original model. Users often choose to copy plans from their own previous projects if it is too difficult to comprehend a model developed by someone else. Unlike primary activation, the people who reuse a model may not have access to the domain (D_1) it originally reflected. Developing and maintaining a library of reusable model templates is one way of approaching this problem.

Metamodelling ($K \rightarrow L$)

Metamodelling involves changing the modelling language. It is typically carried out in order to improve the comprehensibility of the language for human actors, or to make it more suitable in the local domain. Metamodelling faces similar problems as reuse, but may also benefit from the immediate domain availability, which characterises interactive models. Local language adaptations should increase the pragmatic and semantic qualities of the resulting models.

Reflection (L⊂M)

In knowledge-based systems, the operational logic is stored as data rather than programmed in software, while reflective systems expose representations of their own logic to their users, and allow modification of this logic [138, 268, 269]. Interactive models combine behavioural reflection with user interaction through visual models. A modelling framework (M, L) is reflective if the language that is used to build models is itself available and can be adapted as a model. Reflection enables us to view metamodelling as articulation, and reuse as transfer of both the model and the language.

2.1.3 Interactive Models are Socially Constructed

Organisations and models can be viewed as socially constructed realities, as artefacts that help shape people's perception of reality, of themselves and the world they live in. As mentioned earlier, models are defined as representations of some part of reality as perceived by some actor(s). Berger and Luckmann define reality as a *quality*: "*Phenomena that are real have a being independent of our volition*" [43]. In their sociology of knowledge, knowledge is defined as certainty that phenomena are real and that they possess specific characteristics. Social construction of reality refers to the processes whereby something becomes real and accepted knowledge.

The classic definition of the term model creates a dichotomy between what is represented (reality) and the representation (model). A social constructivist perspective sees the model also as reality, if it is accepted as such in the community where it is used. Acceptance as reality is crucial for interactive models. For instance, a process model will not serve its function if it is not accepted as a real depiction of past, present and planned activities. Representations *"become 'dead', (that is, bereft of subjective reality) unless they are ongoingly 'brought to life' in actual human conduct"* [43]. The interplay of model articulation and activation (Figure 2) is thus a process of social construction.

Language

Berger and Luckmann also describe the role of language in social processes. "*As a sign system, language has the quality of objectivity. I encounter language as a facticity external to me and coercive in its effect on me. Language forces me into patterns"* [43]. The coercive effects of language increase with formality and specificity. Languages that limit interpretive freedom [51] and range of expression are likely to be experienced as constraining. This is the case with several information systems, especially those who regard work as structured and well understood. It also offers an explanation why systems that structure conversations have met resistance from users [124, 469]. Systems should thus not require that every part of an interactive model is formalised and interpreted by the computer. It should also allow humans to communicate freely through the models and perform parts of the interpretation themselves.

Subjective and Objective Reality

Approaches to information systems engineering and conceptual modelling have been categorised based on their ontological and epistemological stance [221, 442]. Ontological *realism* assumes that reality exists independent from us, whereas ontological *idealism* (or nominalism) denies that position [442]. In the epistemological dimension it is common to distinguish between *objectivism* (the position that objective knowledge is possible) and *subjectivism*. Seeing reality as socially constructed allows us to connect these perspectives in ways that are especially relevant for interactive models.

Berger and Luckmann [43] integrates objectivity and subjectivity in an ongoing dialectical process of articulation, objectivation and socialisation. Subjective reality affects objective reality through articulation, and is affected by objective reality through socialisation. This interplay is related to that of the domain (subjective realities) and an interactive model (becoming objective reality) depicted in Figure 2. On the other hand, models should also reflect the users' subjective realities, in order to support storytelling [77, 387] and negotiation of meaning [527], the social processes whereby knowledge becomes shared and objective.

Ambiguity is a Prerequisite for Meaning

Within a community of practice [77, 527], social learning takes place through negotiation of meaning. This negotiation requires that people actively and legitimately *participate* in the working practices of the community, but also that they make their perspectives tangible to others through *reification* [527]. Reification involves elucidation of concepts and construction of artefacts embodying perspectives. It thus complements the conceptual focus of articulation with a physical perspective. Interactive models, while primarily conceptual, become tangible in the user interface of the information system.

Participation and reification are intertwined, supporting and complementing each other. Reification only makes sense through participation, and it is a medium for communication and thus participation. Models can be analysed according to how the work of negotiating meaning is distributed between reification and participation. A poem requires much participation to be understood, while a software program does not require human participation in its meaning [527]. The aim of conventional IS models is to reify programs, while interactive models also aim to foster *participation* in an ongoing process of social learning and reality construction.

The duality of reification and participation has bearings on the suitable level of specificity for a model: "*When combined with history, ambiguity is not an absence or a lack of meaning. Rather, it is a condition of negotiability and thus a condition for the very possibility of meaning*" [527]. Ambiguity is a catalyst for participation, because reifications that only allow one interpretation, are not open to negotiation. Interactive models must thus allow ambiguity rather than formalise one "correct" interpretation.

2.2 Articulation of Interactive Models

Interactive modelling is based on the assumption that end users are capable of creating and adapting models of their own work. Several researchers doubt the feasibility of such approaches [35, 45, 213]. This section therefore explores case studies of modelling and articulation, demonstrating the need for interaction, but also uncovering a number of requirements and remaining challenges.

2.2.1 Articulation Work

For interactive models, the term 'articulation' is preferred to 'modelling', 'representation', and 'externalisation' because

- Articulation is a well-established concept in CSCW and applied e.g. in coordination theories [178, 218, 220, 414, 438].
- Articulation is generally discussed as an integral part of work, not as a separate activity, and articulation is itself regarded as a work activity that requires time and effort [438]. This matches the concept of interactive models created and maintained by end users as part of their day-to-day planning and coordination of work.
- Unlike 'modelling', the term does not carry implicit connotations of graphical flowcharting, formalisation, abstraction and generalisation. Interactive models should refer to concrete objects and ideas.

2.2.2 User Participation in Systems Development

Interactive models and system development models differ with respect to who does the modelling. Interactive models are constructed by ordinary users. "*The generation of representations can only be done successfully with the participation of the people who live in the situations being represented*" [537]. In systems analysis, users are often seen more as sources of information than as model builders [221], and even participatory design [150] involves a limited number of user representatives, not the whole community immersed in their day-to-day activities. Hence, interactive models face stronger requirements for language comprehensibility, both because some users initially lack experience with modelling, and because evolution and learning will cause more frequent updates to the models.

The opportunity to rapidly update the IS is one of the main advantages with interactive models. In systems engineering agile, iterative and incremental development [219] attempt to shorten the learning cycles, but they are still hampered with a long time-span from learning to system-change. Users have more in-depth knowledge of their domain than software developers. Consequently, the potential for high semantic quality is greater, provided the language is simple, adaptable and user-oriented.

The basic concepts of an information system are normally defined during development. Most users thus encounter a terminology constructed without their participation. Though engineering methodologies generally highlight the importance of user involvement, it is infeasible to involve all users. The experience that user representatives gain through participation the social construction of the system, cannot be shared by all, even if the representatives help in training and user support.

Users are Active Participants

The core principle of participatory design (PD) is to view users as contributing actors, not just sources of information. The importance of mutual learning for both developers and users is stressed [150, 267, 296]. Within this framework, a number of projects have been carried out that illustrate the feasibility of active user participation. Bansler and Bødker [33] surveyed the use of structured analysis with data flow diagrams. They found that the techniques were applied pragmatically, and that prescribed methods were not followed. The subjects however reported that diagramming was useful, and they would apply them again in future projects.

Criticism of Categorisation

Participatory design has put more emphasis on tangible reifications directly comprehensible for end users, like prototypes, scenarios and mock-ups [150], despite the tendency of these representations to create a focus on superficial user interface concerns [296]. In particular, the use of predefined modelling types has been widely criticised [124]. Suchman [469] argues that encoding of intentions into explicit categories carries with it an agenda of control over workers' actions. Categorisation makes actions more manageable and communicable, while creating a danger of reducing the actions to its category, of hiding aspects of work not captured in the categorisations. Textual scenarios, often created in story-telling workshops, are thus preferred to models [272]. Other researchers have noted that "*integration of diagramming techniques and additional media facilitates understanding and provokes discussions on organisational and technical issues*" [215]. This research indicates that it is easier for users to understand details by looking at screenshot prototypes, but also that users relate their input to the graphical models. With an interactive model, these two media are linked together, complementing each other in an even more direct way.

Intuitive Visual Languages

Other studies of end user participation in IS and enterprise modelling counters the somewhat negative views presented above. Latour [300] shows the tremendous importance of graphical representations, maps, blueprints, and geometrical models, in the advance of science and technology. The intuitive power of iconic representations over pure symbols with no similarity with what they represent has been elaborated [188]. It is also pointed out that this stands in sharp contrast to formal computer science, where new symbols are often introduced which are not used by any community of practitioners, and most often are purely symbolic [188].

Modelling Language Complexity

The modelling conference is a method for participatory development of simple, process-oriented enterprise models [181]. In a consulting firm, this method was applied to get users involved in the development of a corporate Intranet portal. With a tangible modelling interface (yellow stickers on a whiteboard), groups of end users were able to build quite elaborate models. Broad participation in modelling workshops did increase the quality of the resulting models. A related study observed that IDEF0 is too complex for this kind of modelling, and consequently simpler symbols were selected [182]. Flowcharts were useful for the workers to pinpoint important issues, but a need for complementary visualisations of other perspectives was also identified. In one company, the enterprise model was made operational, in that functionality was added to an online model to ensure that it was used [182].

Modelling as an Ongoing Process

The SEEME modelling language is developed to support requirements negotiation, not just rationalistic requirements capture [215]. Thus, SEEME allows users to explicitly represent vagueness, incompleteness and contradictions in their models. In case studies, researchers were surprised by how naturally the users adopted these concepts. While models were originally built by analysts, end users, even those with very limited understanding of the diagrams, were able to add a wide variety of elements and to correct errors in the models. Far-reaching discussions concerning too much sequencing in the process models were triggered. In this situation, the original view of IS professionals did not match that of the users.

Domain-Oriented Modelling

The formal languages applied in IS development have been described as foreign and opaque to users [4]. New designs should evolve not in isolation but by adopting and extending the languages that users are already familiar with. Instead of bringing user representatives into development projects, participatory design in the workplace [509] place developers within the usage context. While this solves some of the problems of user participation, it is infeasible to place developers in every workplace. Methodologies are needed that empower users to control the system without the modelling expert as a mediator. Reflective tools that enable the construction of domain-oriented languages, have experienced some success in this area [492]. Industry reports [345, 346] claim order of magnitude productivity gains, shortened development cycles, and improved learning curves for newcomers.

2.2.3 Innovation by Users during Information Systems Operation

The objective of an IS should not be to satisfy formalised user requirements, as user needs and users' understanding of their needs will evolve [497]. Consequently, "*the key criterion of a system's usability is the extent to which it supports the potential for people who work with it to understand it, to learn and to make changes*" [4]. Facing exceptions, users will not always abandon the system if they are able to change its configuration [437]. A number of case studies report that users do indeed apply IS in ways not intended by the designers, and that many are also capable of ad-hoc modification and integration of tools to meet their particular needs. The following subsections survey some of the most relevant work in this area.

From Sub-Optimisation to Process Orientation

While industrial manufacturing is not typically thought of as knowledge intensive work, automation is changing the nature of workers' responsibilities towards more planning, monitoring and exception handling [547]. One study reported that workers usually optimise their own activities, but that they have neither the tools nor the knowledge of what goes on earlier or later in the process [182]. Increased visibility and process orientation is needed to avoid sub-optimisation. Experiences with user-led development during the operation of the system, shows that this is feasible [509]. Various exceptions caused workers to increase their *horizon of visibility* to include other parts of the business processes. Workers also maintained a local complaint book where all issues regarding the plant and its information systems could be collected. The rapid pace of product innovation has caused some factories to de-automate, replacing robots with human workers, because their intelligence and flexibility is needed, e.g. in mobile phones production [122] and integrated car manufacturing [273]. Hence, user involvement in manufacturing process improvement is feasible, provided appropriate tools are available.

Case Processing is also Knowledge Intensive

Case processing is often portrayed as the new factory work. Case processing industries are among the key customers of workflow management [167, 187, 532]. Following these perspectives, this work should be routine, and not require specialised skills. When researchers look behind the scenes, however, they discover a rich variety of knowledge intensive casework [387, 527]. Wenger [527] shows the importance of informal social networks for dissemination of knowledge in insurance claims processing. Systems support this by allowing informal notes to be added to cases. Local jargon and sophisticated domain terminology permeate the workplace. While newcomers are initially trained in processing claims "the right way", it is informally understood by all that they need to learn shortcuts to get the job done. Like Orr's study of copier repair [387], Wenger shows that even seemingly unskilled people ongoingly reflect on their work processes. The social features of learning, mediated by storytelling and problem-oriented inquiry, is also common to these studies. Another study of seemingly routine computer-supported work [176] observed that people knowingly input false data to obtain correct results, and that manual systems are constructed as workarounds for inflexible computerised IS.

Alienating Technologies

Zuboff [547] outlines the history of clerical work, highlighting how principles of scientific management [485] created a focus on simplifying routines, automating coordination, and removing the need for communication. These theories have also influenced workflow management [92]. A case study in the 1980s, reported that caseworkers felt increasingly alienated from their work [547]. It had become impossible for them to put their personal stamp on the interaction with the customer, and they wondered where the material on their computer screens came from and where it went. The later study by Wenger that we discussed above shows that with more user-friendly and flexible IS, users share knowledge and innovate practices. Systems should thus be designed for innovation and empowerment.

Rationalisation and Multiple Points of View

Notions of office work that emphasise routine, canonical activities miss the rich meaning of social action in an office [220]. By representing work as rational, deterministic and overt, important political, cultural and cognitive problems are overlooked. From this perspective Gerson and Star [178] propose that more emphasis should be put on local, tacit knowledge and its transferability. When problems have no globally correct answers, multiple competing and perhaps incommensurable proposals develop. Articulation work is the social activity directed at reconciling incommensurate assumptions and procedures [178]. In their case study, Gerson and Star identify nine groups with different viewpoints, and describe how they interact in the changing of procedure definitions, insurance payments, fees and coverage rules. All viewpoints contribute to a continuous evolution of the company's practice, and every single case involves a new interpretation. No formal description of this practice can ever be complete, and every real world system requires articulation to deal with unanticipated contingencies. It is also pointed out that no piece of real information is simple, that categories always have a complex context and history [178].

Tailoring Information Systems and Changing Organisational Practices

Flexibility has many aspects. It involves personalisation and local modifications as well as far-reaching evolution of information systems. Flexibility is a requirement for several components, from shallow adaptability of the user interfaces to deeper reconfiguration and extension of functionality. Available hardware and software infrastructures change all the time, constituting yet another evolution challenge. Tailoring is defined as adaptation performed by end users as part of normal system use [488]. Interactive models are visual domain representations for tailoring.

Studies of human computer interaction have established that people are capable of utilising a wide range of customisation services when the IS offer them [259, 363, 493]. Through informal collaboration, customisations spread from user to user in the organisations [362], adding a knowledge management perspective to tailoring. Trigg and Bødker [493] notice that users tend to modify and extend their work practices alongside tailoring the information systems. Their study shows how word processor users frequently start by copying an existing document, reusing its structure and formatting. At first, forms, macros etc. were developed and spread informally, but as the value of these activities became evident, a more structured process with defined roles was put in place. Studies of general-purpose hypermedia systems have also observed users building form templates [452]. Noticing the difficulties users have with predefined formalisations, approaches for user-definable *incremental formalisations* are developed [453, 454]. Experiments and case studies show many examples of incremental formalisation [451].

User Participation in Knowledge Management

An important motivation for interactive models, and in particular for model reuse, is the need to support organisational learning and knowledge management. A number of different tools have been utilised for knowledge management [64, 258, 325]. Studies of user participation in these systems are highly relevant for assessing the feasibility of interactive model articulation by end users.

One study shows the importance of knowledge connectivity (e.g. models with many relationships), user connectivity, contextual dependence, and user idiosyncrasies for knowledge capture, retrieval and discovery [344, 510]. Another noted that computerised information systems lack the idea generation capability and serendipity of personal, face-to-face conversations [343]. On the other hand, Clarke and Cooper [106] show that introduction of intranets for knowledge distribution often is accompanied by the emergence of new collaborative organisational structures, called 'networks', 'teams' or 'communities'. The technology creates an opportunity for increased interaction, and users seize that opportunity. Usefulness and relevance as perceived by end users, was identified as the most important success factor for organisational memory systems [13].

A case study of the adoption and adaptation of groupware support for a virtual team [324] shows both the potentials and limitations of using information systems to transform organisational practice [307]. The technology in this case was an Internet-based asynchronous groupware tool that provided a shared repository, with event notification, conversation support, search and navigation services. In spite of commitment by all parties involved, there were many barriers to changing work practice. Initially the group tried to adjust their working practices to meet the vision of the technology, then the focus shifted towards adapting the system to support a workable practice. Interestingly, the one aspect that really did change was the group structures for communication and decision making. A participative, interdisciplinary culture developed, where all project members discussed aspects of each other's tasks. This led the group to question some of the original requirements, which resulted in breakthrough innovations. In complexity, reliability, and price the result represented orders-of-magnitude improvement.

Summary

The case studies surveyed in this section show that even seemingly unskilled workers reflect sophisticatedly upon their work. Reflection is an integrated part of practice. This evidence prompts us to view 'knowledge intensive work' more as a *perspective* on work, than as a category of work. It is also evident that user communities discuss and share their reflections, informally disseminating knowledge. Some studies show examples where information systems enable user innovations to spread across the organisation in an opportunistic manner. Other studies show systems that prevent local innovations and informal communication. As a response to these systems, social networks arise to humanise the workplace and maintain the smooth flow of work. It seems that the scepticism some engineers have towards the feasibility of end user control [35, 45, 213] becomes self-fulfilling. Systems that are not designed to support tailoring create barriers to innovation and learning. The usability of visual models is also disputed. While some studies show barriers to end user participation in model articulation, others observe sophisticated structures being created ad-hoc to serve local purposes. There is also a large body of research that identifies weaknesses and limitations of formal modelling techniques, and the severe consequences of building overly simplistic assumptions about the nature of work into the systems.

2.2.4 Requirements for Articulation Support

Articulation depends upon subjective comprehensibility and perceived utility of the models and the modelling language. The requirement that languages should be formal [115] must thus be balanced with a more pragmatic view. Ambiguity and uncertainty are vehicles for professional learning, triggering open discussion of what the models mean. This view of models as socially constructed in communities of practice, generates a number of particular requirements for modelling languages:

R1: The language should be *simple*, with few basic concepts [7, 241]. We should avoid overly detailed categorisations [254]. It should be straight-forward to determine what a model element represents. Concrete examples provide common ground for negotiation of meaning. Each real world object should have one and only one model element representing it [385].

R2: The language should allow a *visual*, *graphical* depiction of the model, giving both an overview of the whole process and details about its parts [89, 188, 241]. Multiple dynamic views that extract the aspects of the model most interesting to a particular purpose or role should be available [241, 254, 418].

R3: Language constructs must map well to the conceptual world of those performing the work [200, 254, 417]. *Domain specific* concepts should be utilised [399, 492].

R4: The meaning of a model element should not always be fixed. It should be able to evolve as the negotiation of meaning unfolds [81]. Participants should be invited to reflect upon their language and their process of negotiating meaning [527].

Articulation need not entail visual modelling. Any interface that allows users to alter data is an articulation tool. One example is a spreadsheet that lists a task hierarchy with responsible persons, deadlines and resource consumption. This spreadsheet is a process model. In some cases, such textual interfaces correspond more directly to the way people currently plan their projects.

2.3 Activation

While activation depends on the richness, detail and precision of the articulated representations, is also motivates articulation by increasing the users' benefits of keeping the models up to date. Compared to the rich literature on modelling, languages and representation, the research on activation is limited. Directing the focus towards activation is thus a major contribution of interactive modelling. In specialised areas like ontologies, workflow, and product data management, model activation mechanisms are developed for deduction [502], enactment [143] and cooperation support [163, 233]. Conceptual modelling languages have been made executable in order to support verification and validation during systems development [535]. Recently, virtual machines for the activation of Unified Modelling Language (UML) models have emerged as a research area [411]. Most previous work however focuses on formal execution semantics for closed systems. "*It is usual for computational design methodologies to adopt a naively realist stance toward their ontological analysis of human activities, as though the resulting formal structures could be read straight onto the structures of human action without the mediation of human action or interpretation*" [9].

2.3.1 Informating and Automating

Zuboff [547] shows how information systems are used both to increase learning and the capabilities of the workers, by *informating* them, as well as to *automate* local processes and centralise decision-making. This duality is reflected in the separation between automatic and manual model activation. Manual activation requires comprehensible, flexible and user-oriented models [188, 217, 452]. Automation demands formally defined operational semantics, and a complete model. Interactive activation must balance these conflicting requirements, allowing users to switch between automation and manual control depending on which best fits the situation at hand.

2.3.2 Degrees of Formality

The essence of IS design is the reconciliation of social and technical aspects [190]. The gap between system-executable formalisms and domain languages is not easily bridged. Using different sub-languages for automated and manual parts makes it difficult for users to grasp the combined behaviour, and difficult to move between ambiguity and formality. Interactive modelling languages should thus integrate different degrees of formality [16, 45, 116, 326]. Goguen [189, 190] apply a metaphor of *humidity* to discuss the formality of a language, ranging from formal (dry) to informal (wet). As he points out, information cannot be fully context sensitive (wet), for then it is only understood when and where it is produced. Nor can it be fully context insensitive (dry, formal), for then it could be understood by anyone anywhere at any time. In this framework, formalisation is defined as the process of making information drier and less situated. The limitations of formalisation include cognitive costs of learning the language, effort required to make the model complete, conceptual gap between users' knowledge and the system, enforcing premature structure before the group process has reached that far, and poor support for situated social activities [338, 452]. Experience indicates that "*features requiring greater degrees of formality end up being less frequently used*" [452].

2.3.3 Open and Closed Systems

Closed systems need modelling languages with formal semantics. When a model cannot be elaborated during its execution, it must be predefined in a formally complete manner, encoding all relevant details from the start. This perspective treats *process models as programs* [389] and *systems as mechanisms* [41]. Open systems that support sharing of model information between users, need not require a formal language. Here models are treated as data and *systems as media* for communication and collaboration [41].

Incompleteness or Complexity

Interactive activation allows the input of model data to be delayed right up to the time when that part of the model is executed. When the model does not tell the system what should happen next, users are asked to decide. Consequently, the model may remain incomplete throughout its lifecycle, and alternative paths that are not taken, need not be defined. This simplifies the model. Research on formalising dynamic workflow models show that ambiguity and flexibility can lead to exponential growth in model size when all alternatives must be represented [7, 549]. This exponential growth makes the models hard to comprehend. Where models in closed systems must define all *potential* patterns, interactive models need only include the *actually* occurring patterns.

Uncertainty is Inevitable

Some degree of uncertainty is fundamental to social activities and our making sense of them. It is thus recommended *"that we turn our focus from explaining away uncertainty in the interpretation of action to identifying the resources by which the inevitable uncertainty is managed". "Interpreting the significance of action is an essentially collaborative activity"* [468].

Variation, Evolution and Reuse

Formal semantics simplify reuse. Once a complete model exists, it can be instantiated over and over again. Large volumes and high costs may make the initial effort of detailed formal analysis viable for routine, repetitive process. The more details are put into the models, the less information is required as input in each individual case. However, the rate of change and frequency of exceptions may make the model out of date, requiring evolution. The need for personalisation, customisation and motivation through local ownership, as opposed to alienation due to incomprehensible systems [547], further indicate that the open systems approach is better suited for a wide range of work processes.

Organic Growth

An open system with interactive models allows IS to be grown organically [75, 497]. Starting with a rudimentary model of the main steps in a process, repeated executions of similar processes capture different ad-hoc extensions and decisions by end users. By combining several incrementally constructed process instances, a more general and complete model emerges. This scheme entails immediate benefits without large initial investment, and matches well an incremental, need and risk driven software lifecycle [127].

2.3.4 Change and Evolution

Ability to change is an important competitive factor in most industries. Customers' requirements change frequently and vary from case to case, creating a need for solutions tailored to each client. Consequently, who your competitors and partners are, change. We thus need systems that treat change as the rule of the game, not as exceptional. Management of change is a main challenge for information systems [123]. In the terminology defined in section 2.1, some changes follow the dynamics of the model, while others require that the model evolve. Models evolve at two levels [550]:

1. *Local* modifications to the running model, leaving the generic definition unchanged.
2. *General* and long-lasting modifications to the generic definition, so that ongoing and future instances are affected.

The degree of evolution required is also commonly used to distinguish among classes of exceptions. Kammer et al. [261] distinguish among noise (not articulated in the models), idiosyncratic (instance level) and evolutionary (class level) exceptions. Of course, not all systems allow modifications. Workflow management systems seldom allow local modifications, hardwiring process definition to the class level [532]. For interactive models, on the other hand, this creates research issues in determining when updates made to a running instance should be migrated to the generic level. This is a challenge for model harvesting. During activation, we must handle both local modifications and dynamic change (updates to local models caused by evolution of general models [153]).

Exception Handling

In closed systems, change is regarded as exceptions to predefined rules [103, 275, 320, 464]. An *expected* exception can be represented in a predefined model, but it captures a deviation from the normal or desired course of events. Building excessive details of exception handling into the model makes it complicated, hard to comprehend and hard to change. An *unexpected* exception is caused by a change in the system's domain that was not anticipated at modelling time. Such exceptions require the closed system to be opened up, to allow exception handling by users, model evolution, or the case to be taken out of the system and handled manually [69, 140]. Strong and Miller [464] analyse different perspectives on exceptions and how they should be handled. The underlying assumptions include

- Exceptions as unpredictable, random events,
- Exceptions as errors, as indicators of underlying problems,
- Exceptions as normal, as a part of necessary process flexibility.

Their case study indicates a need to move from manual intervention as a strategy for exception handling in an otherwise automated process, to an interactive process supervised by humans. They advocate designing for evolution. Interactive models enable exceptions to be built in right up to the time when the model fragment is activated. Hence a greater number of exceptions can be moved from the unexpected to the expected category, due to the learning of the participants. By not requiring formally complete and consistent representations, a wider range of exceptions can also be tolerated. The system need not break down in the face of exceptions, it can just offer its best effort at interpreting the model in the new situation, and rely on human actors to adjust the interpretation if needed. This allows user organisations to decide the degree of exception handling detail that should be articulated for each process.

Personalisation, Customisation and Ownership

Cooperative knowledge work relies heavily on the intangible skills of individuals. Organisational theories thus advocate decentralised decision-making and empowerment. Empowerment requires end-user customisation of system features, enabling them to create personalised workspaces. Personalisation fosters a sense of ownership. Ownership, autonomy and responsibility motivate people. *Appropriation*, the process where a general template is adapted and adopted into a particular work setting, converted to a locally owned object, is thus a key challenge in model reuse. In addition to personal preferences, cooperative knowledge work requires solutions that can be adapted to the situated needs of the project community. Since interactive models capture important aspects of this local setting, they are a vital resource for customisation. We refer to model-driven customisation as *contextualisation*. Contextualisation depends on the semantic quality of the model (how accurately it represents the domain). Local model evolution is critical to achieve this, and improving the contextualisation capabilities of modelling languages, is an important research challenge.

2.3.5 Increasing the Functionality While Avoiding Featuritis

Functional features sell systems and motivate use, but the complexity of choice among a wide range of specialised functionality often becomes a severe usability problem. Similarly, formal scientific research into process modelling emphasises expressiveness, often at the expense of simplicity. The *efficiency* of a language is defined as its expressiveness divided by its size (the number of primitive constructs) [82, 189]. Language efficiency is increased if language elements can mean different things in different contexts [189]. Most approaches to model activation, e.g. in process support systems, apply each language element for one specific purpose. Attempts at increasing the usefulness of such systems have added conversation support [6, 124, 233], awareness [415], information management [365], and other groupware services [16]. Most often, integration of additional functionality is accomplished by introducing new modelling constructs for each service. This complicates the system. In order to improve language efficiency, simplicity and usability, different perspectives should be combined within a single language.

2.3.6 Emergent Interoperability

When interorganisational cooperation moves beyond the buying and selling of well-defined goods and services, there is a need for a flexible infrastructure that supports not only information exchange, but also knowledge dissemination. We must be able to form effective teams across organisational boundaries and local cultures. Also, the ability of each organisation to learn from the experiences of the joint enterprise is crucial for long term success. The transient and situated nature of each project, and the large and shifting set of partners, seldom makes it economically viable to integrate information systems through normal development practices. Standardisation is often proposed, but require that the domain is static and well understood. This is seldom the case for knowledge intensive work. Consequently, we need a flexible infrastructure that allows shared understanding, process integration, semantic and technical interoperability to *emerge* from the project, rather than being a prerequisite for cooperation.

2.3.7 Requirements for Activation Support

The value of an interactive model lies in the way it affects practice. During the performance of work, models can be utilised for automation and for *guidance* [143, 247]. Interactive activation combines the two. Viewing learning as a social process intertwined in practice, our focus includes the evolution of models during activation, as users adapt them to handle exceptions and changes, and to incorporate new experience.

R5: The language should enable *local change* to a particular model [115, 117, 417] to match the local circumstances, preferences and knowledge of the users. The history of events and model changes should be accurately captured, including exceptions and violations of standard definitions [117, 247, 254]. This history is important both for learning and for contextualising the activation mechanisms.

If the information system offers flexible support based on the current state of the process model, the users will benefit more immediately from keeping the model up to date. If these benefits are less immediate, e.g. input to a separate knowledge management activity performed by someone else, end user motivation may suffer. Participants will always have *tacit* skills and knowledge that is not externalised in the models [370]. It is therefore unlikely that a knowledge intensive work can ever be completely prescribed, rather

R6: The system must be able to interpret and activate *incomplete* models [58, 143]. It must allow ambiguity and not enforce premature formalisation [452]. One should not require models to be predefined, it should be possible to articulate models from scratch as part of the work [308, 451, 493].

R7: Information systems should be *contextual* and *personalisable* [6, 143]. Models should be utilised to provide contextual support, and users must be supported in overriding predefined rules when the situation requires it.

2.4 Reuse and Organisational Learning

Interactive models aim to facilitate more productive and innovative behaviour in the organisation. This requires learning. In the past, models have been used in quality control manuals and operating procedures. Often these documents end up in dusty binders and are seldom activated. If models are to be a useful learning tool, they must relate to the natural learning processes in an organisation, integrated in everyday work. What people need is "*not the partial, rigid models of the sort directive documentation provides, but help to build, ad hoc and collaboratively, robust models that do justice to particular difficulties in which they find themselves*" [77].

2.4.1 Practice and How We Describe It

Recent studies indicate that the ways people actually work differ fundamentally from the ways organisations describe it [77, 387, 468]. Concepts like theories-in-use and theories-espoused [19, 20], and talk, hypocrisy and action [80] refer to the fact that *what people say they do* differs from *what they really do*. Case studies have shown that this gap is a major barrier to organisational learning [19, 20]. Methods that create models of work as input to design, have been criticised for biasing organisational, explicit views and for obscuring the communicative practices, skills and knowledge that are crucial to efficient working [31, 471]. Suchman [471] claims that "*current wisdom in system design holds that the less of a user's behaviour a system encodes, the less functionality it can provide*". Interactive models offer a different perspective, enabling users to externalise and communicate about their practice, rather than encoding canonical models (official accounts of work [77]) in software. This approach thus offers a great opportunity for knowledge management anchored in practice.

The *immediateness* of interactive models enhances their social pragmatic quality. When both the real world and the model that reflects it are available and adaptable, the connections between them are easier to understand. Simulation and training methods can be developed that utilise this connection. Zuboff's study of industrial control rooms [547] shows great benefits for users that are able to work both with the conceptual tools and the physical environment of the factory. On the other hand, the study also highlights the pitfalls of systems that isolate users inside the modelled world of the control rooms without understanding what really goes on in the plant. Whenever a user discovers an anomaly in an interactive model, it should be easy to correct it. This implies that model-activating software components also should integrate articulation support.

Reflective Practice

Schön [445] describes professional work as reflection in action (cf. Appendix A.2.1), integrating problem setting and problem solving in an ongoing conversation with the current situation. Each new situation is treated as a *unique whole*. The wholeness of a problem is not subsumed under categories, but features and elements of the situation can be categorised [445]. This is important for understanding how knowledge developed in one situation can be used in a different setting. Similarly, Brown and Duguid [77] note that learning theories have "*rejected transfer models that isolate knowledge from practice and developed a view of learning as social construction, putting knowledge back in the contexts where it has meaning*". Schön proposes that practitioners build up *repertoires* of examples, images, understandings and actions. The practitioner makes sense of a new situation through seeing similarities with patterns in his repertoire, by using familiar situations as precedents or metaphors for the new, unique problem [445]. In addition to personal experience, the repertoire can include stories and examples that we have read or heard about [77, 387].

Language of Practice

Communication in reflective practice employs the language of design (for reflection-in-action) but also the language *about* designing (for reflection-on-action). Thus, if interactive models are to be useful for reflective practice they must themselves be reflective; they must facilitate conversation about the language, the work process and the problem setting. As pointed out by Berger and Luckmann: *"Language also typifies experiences, allowing me to subsume them under broad categories in terms of which they have meaning not only to myself but to my fellowmen." "Language becomes the depository of a large aggregate of collective sedimentations, which can be acquired monothetically, that is, as cohesive wholes and without reconstructing their original process of formation".* [43]. Language helps us categorise and attribute meaning to our experience, in short, to reflect on our actions. This property is crucial for learning from experience, and creates the capability to transcend "here and now". Hence, the social processes of language construction should be supported, and local language variants should be treated as important sources of knowledge.

Models of Practice

Schön's model of reflective action offers fundamental insights into the processes where models are constructed. Models represent problem setting descriptions, but also fragments and aspects of solutions. Model articulation can thus be viewed as a conversation with the situation. In this conversation each step is an experimental move that can bring the practitioners closer to an understanding and a framing of the problem, and possibly contribute to an acceptable solution. The knowledge that is available in this process includes the repertoires of actions possessed by the participants. Through harvesting and reuse, interactive models become part of an explicit repertoire of action for the organisation. Model reuse is essential because it decreases the effort of modelling and increases the potential value of each model. The anomalies and creative tension that comes from using an old model as a metaphor for your current situation can foster learning and innovation [370].

2.4.2 Communities of Practice

Reflective practice describes how individual professionals learn and solve problems. Community of practice is a *social* theory of learning, where learning is regarded an inherent part of everyday interaction with others [77, 527]. Wenger [527] outlines this theory from four interrelated perspectives:

- *Meaning*, our changing ability to make sense of our life and the world.
- *Practice*, shared historical and social resources, frameworks and perspectives that can sustain mutual engagement in action.
- *Community*, the social configurations in which our enterprises are defined as worth pursuing and where our participation is recognisable as competence.
- *Identity*, how learning changes who we are and creates personal histories of becoming in the context of communities.

From these perspectives, learning is viewed as experience, as doing, as belonging and as becoming. Shared understanding is socially constructed inside a community through *perspective making* [60]. Across communities, peripheral participants, multi-community membership and *boundary objects* ([462], cf. Appendix A.1.3) enable learning through *perspective taking* [60]. Boundary objects are ambiguous to foster participation in negotiation of meaning inside a community, but maintain a clear identity so they can be transferred across communities. They thus become structured through use, assigned different, but overlapping, meanings in different communities.

In systems development, software engineers commonly lack a history of participation in the environments where the system is intended to function. Requirement specifications are boundary objects that often emphasise formal reification over ambiguity and participation (cf. section 2.1.3). Participatory design (PD) thus trains developers in the work that their systems are to support. PD has not been completely successful. This can be attributed to political struggles and resistance of interest groups, but other weaknesses should not be ignored. Among them are large initial costs, difficulty in participating in another culture and semantic field [43], and downplaying the role of the software professional as a reflective practitioner [337, 445]. Also, viewing learning as "*becoming* *a practitioner, not learning about practice*" [77], user participation in software development and developers participation in the users' practice can never fully enable them to understand each others' reality.

In contrast, interactive models delay decisions from design-time to runtime, and aim to foster participation of whole user communities in adapting the system to fit local practice. "*The experience of end users cannot be effectively mediated by representation of work or representatives of users, such as systems analysts or ethnographers. By such mediation, experience with the work being supported is frozen at the level of explicit understandings existing at the time when the representation of work is made or when the user-proxy finishes his or her analysis*" [296]. It can thus be argued that interactive models have greater potential for producing user-friendly systems, but also that design and usage of such systems is more challenging than for closed systems. Barriers to participation will be discussed in section 2.4.4 below.

2.4.3 Models and Tacit Knowledge

Theories of knowledge creation and dissemination are highly relevant for the application of interactive models to support learning and innovation. Models are a structured kind of explicit knowledge, created through externalisation [370], and maintained with a rich background of tacit knowledge [39, 402]. Tacit knowledge encompasses informal skills and cognitive mental models, beliefs and perceptions, which have not been made explicit in language. In this section we look at how interactive models can facilitate spirals of knowledge creation [370], as shown in Figure 7.

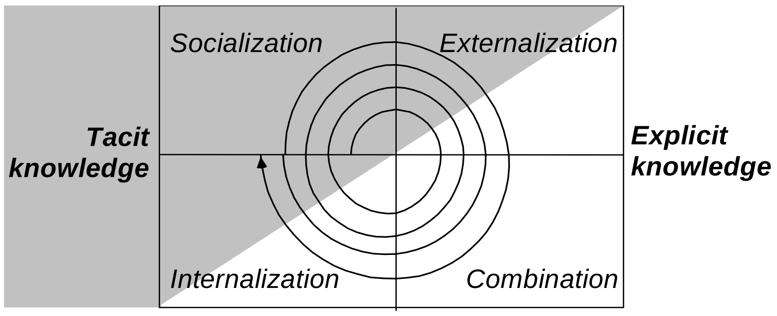


Figure 7. Knowledge creation spiral [370].

Externalising Interactive Models

Articulation of tacit knowledge into interactive models was discussed above. Key questions include what aspects of work it makes sense to put into a model, at what degree of specificity. Answers depend largely on the context. "*Knowing refers to what is socially defined as reality, not some extra-social criteria of cognitive validity*" [43]. Externalisation can have negative consequences. The danger of misuse during power struggles is discussed below. There is also the problem of *trivialisation* [43]. When something becomes explicit and accepted as part of the objective reality, it is often no longer the focus of discussion and negotiation. As reported by [39], it can have disastrous effects if important problems become accepted without being properly dealt with. This is the opposite of how ambiguity, fluctuation, and creative chaos [370] supports learning and innovation in communities of practice. Further research is needed to explore what kind of externalisation is beneficial for learning in which contexts.

Combining Interactive Models

Combination refers to manipulation, composition, comparison, categorisation etc. of explicit knowledge. This is the domain that information systems target, and where they probably have the greatest potential. A technology for model-based knowledge management should satisfy these requirements:

R8: Generalisation of models should facilitate comparative analysis [241], and cost-effective metrics [417].

R9: Languages should enable intuitive structuring of large model repositories [329, 541, 550]. The structures should be evolving, adjusting to improved understanding by users, and capable of handling multiple viewpoints and dimensions of models [461].

R10: Modelling languages should facilitate composition of models from parts [92, 516].

Nonaka and Takeuchi however caution designers against narrowly focusing on this mode of knowledge conversion [370].

Internalising Interactive Models

Internalisation refers to conversion of explicit representations into tacit skills and capabilities by groups and individuals [370]. If models are to be internalised, they must, as all secondary realities, be vivid, relevant and interesting [43]. Relevance can be achieved if the IS actively interprets the model to provide the information that is needed in the current context. An interactive model can be vivid if the users see how changes to the model affect the behaviour of the IS, and how they can use this capability to customise the system to their own preferences and local reality. Interest is created when the system answers to real needs of its users in their everyday work, when they are the ones that benefit from the introduction of the system [202].

Socialisation around Interactive Models

Social learning is facilitated by access to and membership of a target community [77]. Socialisation [43, 370] with people who are experts and have experience in exercising the skills one needs to acquire, is often more important for knowledge dissemination than explicit models. This does not mean that models have no role in facilitating socialisation. Indeed, as user-created representations, interactive models can help us identify the people who can give us a richer description of a particular problem. Hence, interactive models can trigger storytelling [77, 387]. To do so, models must identify the users that constructed them. To some extent, a personal touch to models and local language constructs can also mediate the shared knowledge of a community. A *shared action context* [547] for modelling can contribute to socialisation around the models. Such a context is most easily established when modellers are in the same room and manipulate the models together. Distributed action contexts also exist that are at least partially shared, enabled by synchronous conversation support integrated into the modelling tools [233]. In order to facilitate rich knowledge dissemination, the action context of a model must to some degree be recreated in the new setting:

R11: A model should also capture the context and situation where it is suitable, e.g. objectives, assumptions and resource requirements [146, 329]. The system should maintain a link from each template to the instance(s) it was generalised from in order to facilitate learning from examples [445].

2.4.4 Trust, Power and Participation

Trust and openness are crucial for organisational learning. According to Argyris and Schön [20], most of the inhibitory loops that cause dysfunctional organisational learning systems are related to unilateral withholding of information. Senge [447] describes participative and reflective openness as core features of a learning organisation. Participative openness refers to willingness to speak your mind, while reflective openness concerns the willingness to challenge your own interpretations. Information systems increase transparency in the organisation [547]. Still, we cannot assume that information circulates freely just because the technology to support circulation is available [77]. This section explores barriers to participation in interactive modelling.

Legitimisation and Motivation

That an activity is legitimate and accepted by the organisation is crucial for social learning [77, 527]. Thus management support is a commonly cited success factor for the introduction of information systems [202]. Explicit management support is however not sufficient. Attempts at open door management have failed because employees came to realise that though the door is open, it is wiser not to cross the threshold [77]. It is thus necessary to develop a richer view of legitimisation. According to Berger and Luckmann [43], legitimisation means integration of the meaning attached to institutional processes to make them objectively *available* and subjectively *plausible*. Communities of practice theories also demand both legitimisation and availability (*legitimate peripheral participation* [77, 527]).

Withdrawal Strategies

Problems of legitimisation can occur with information systems, if increased transparency is applied to centralise control [547]. Increased outside control can make the workers take to protective measures, like withholding effort [334], and forming informal collectives with norms of behaviour that limit productivity [321]. Such actions have devastating effects on an IS that rely on people's willingness to model and share their insights. There are however indications that these difficulties are diminishing [88]. With increased education and job security, self-actualisation, self-esteem and social needs replace safety and security as main motivational forces according to Maslow's hierarchy [264]. Representing your thoughts and ideas in models, and seeing other people use your input in their work, can be a way of fulfilling these needs. Case studies offer a multitude of examples of this kind of motivation [547]. Education, job security and a sense of autonomy and personal mastery [447] are thus important success factors for interactive models.

Accountability

Another motivation for externalising your work into a model can be to avoid unrealistic workloads. One example is *accountability* as a reason for adopting workflow management systems in healthcare [34]. The system keeps track of activities, documenting that the work has been performed according to the rules [69, 140, 468], but also reflecting the current workload. By pointing to a long list of scheduled work, individuals can demonstrate e.g. that they are unable to handle more cases.

Social Integration

The degree of social integration between different occupational groups is an important part of the societal context of an information system. Studies point to equality as a central condition for identification with a group and formation of communities [321]. Community membership is essential for social learning [77, 527]. In a discussion of copy machine repairers' communities, it is noted that their relationships are "*surprisingly egaliterian*" [77]. Zuboff [547] reports from insurance claims processing that "a *new gulf had opened up between the clerks and their supervisors*", as a result of information technology that replaced mutual problem solving and information sharing with automated coordination of simple steps. The long-term effects of technology-induced social disintegration can be devastating, both on community engagement and production efficiency, as studies of coal mining show [496]. Thus, information technology is both influencing and influenced by the degree of social integration in the organisation. This is a factor that systems designers must pay attention to.

Trust in the Models, Facilitated by Experimentation and Redundancy

Zuboff [547] also discusses the problem of trusting symbolic representations. In her studies, the disjuncture between symbols and actual experience was at first profound, but with time the linkage between the physical and symbolic worlds became tightly wrought. This problem thus seems to be temporary during system introduction, but still it can be crucially damaging for a cooperative system if it prevents reaching a critical mass of users. Reaching critical mass is a core challenge because the benefits of using a cooperative system increase with the number of users [202].

Trusting symbols that reflect abstract functions and systemic relationships is even more difficult [188, 547]. According to Zuboff's informants, people will only trust a machine when they really know how it works. Interactive models enable transparency of the information system, but understanding requires more than just availability. The establishment of training arenas, where users can experiment with the system without interfering with actual operation, is thus a critical measure. To design in *redundancy* is another useful technique. Redundancy allows users to double-check what the system is telling them. It is also important that the impersonality of electronic texts is overcome, e.g. by letting users add informal comments to the model elements both as a personal stamp and as a way of communicating knowledge that they are unable to articulate in the structured modelling language.

Mobility and Physical Barriers to Participation

Zuboff's [547] analysis shows how information technology can lead to the replacement of bodily skills with intellectual ones. Experienced paper plant workers "*were at first overwhelmed with the feeling that they could not see or touch their work*" [547] from inside computerised control rooms. Mobility has also been reported as a limiting factor on knowledge sharing in consulting firms [343]. Interactive models in mobile devices can integrate the concrete world with the abstract data. If a model has a spatial dimension, and its elements are mapped to locations, the system can highlight the elements that are close to the user's current position. Sensors or positioning systems (automated articulation support) can notify handheld devices about the user's position. Users can then combine the rich bodily knowledge they get from sense, sounds and smells with contextual model information, available in the field. If sensors send input to the models, we could create a highly interactive system where users can manipulate the physical artefacts and immediately see hidden and distant effects through how the model changes.

Power and Knowledge

Issues of power and knowledge underlie this discussion. The preservation of managerial authority [519] often becomes an implicit goal of its own, and can be a key force behind automation strategies. As one of Zuboff's informants (a manager) put it: "*If we can build intelligence into our controls, we will not be as dependent on the special knowledge of our crew leaders*" [547]. Employees can and do choose withdrawal strategies when faced with increased managerial control of their work. This does not mean that IS designers should not assume responsibility for whose interests they serve, as more subtle forms of power still can turn their systems into instruments of domination [357]. Suchman thus views representations of work "*as interpretations in the service of particular interests and purposes, created by actors specifically positioned with respect to the work represented*" [470]. Researchers must also pay attention to the 2nd and 3rd dimensions of power in Lukes' framework [107]: Forces that restrict people from representing their interests, and forces that influence on their interpretation of situations, their thoughts and desires. The possibilities that interactive models create for strategic action [300], to enrol others and to control their behaviour, is a key question here.

Language Definition Power, Pluralism and Innovation

The definition of the modelling language is one arena for such actions. Language influences how people in the organisation talk and think about phenomena. Conflicting interests surface, e.g. when a company is seeking to define a small set of keywords for its Internet site. In a hierarchy of terms "everybody" wants their keyword at the top level. Language can contribute to turning certain points of view into implicit, objective facts, and to render other aspects undiscussible. Here, the language becomes part of an *ideology*, a particular definition of reality attached to a concrete power interest [43]. Many people may not have the necessary skills to define their own languages, but language evolution should be supported, because "*pluralism encourages both scepticism and innovation, and is thus inherently subversive of the taken-for-granted status quo*" [43]. System developers, being a social actor in the organisation, cannot completely free themselves from questions about which interests they serve, thus

R12: Modelling languages should be *extensible* so that a community can co-construct its own local dialect as part of the ongoing collaborative learning [417, 453, 451].

R13: Modelling languages should allow multiple perspectives and conflicting interpretations to coexist and evolve [92, 204, 461], and the process of reconciling these perspectives into a common understanding, should be supported and captured [445]. Different viewpoints on each concept should be represented together, not as disintegrated model elements. This is called *perspective integration* [385].

2.5 Process Models

So far we have investigated challenges and requirements for interactive models in general. We now turn our focus to models reflecting work processes. The research surveyed below highlights the importance of process models and process knowledge. The most general definition of *process* just describes a forward motion, a course of events over time [520]. More specifically, 'process' designates "*a method of operation in the production of something*" [520]. Davenport expands on the latter perspective by defining 'process' as "*a structured, measured set of activities designed to produce a specified output for a particular customer or market*" [121]. He further emphasises that process deals with *how* work is done, not *what* is produced. Several perspectives discussed above challenge this definition. Social activities are polymotivated, influenced by multiple goals. Different actors have different objectives, partially understood, partially explicit, partially aligned and partially in conflict. We have also seen that the structure of activities is evolving and locally constructed, rather than predefined. Even with these adjustments, process remains a goal-directed, operational and integrating view on activities.

2.5.1 The Importance of Processes

Organisational routines or ways of working have been described as the identifying characteristic of an organisation [367]. Similarly, theories of social reality construction hold that every institution has a body of transmitted recipe knowledge [43]. Connected to these routines are *roles* that represent the institutional order. Routines are created and changed through social processes, and they are central to organisational development [307] and learning [19, 20, 444]. Nelson and Winter [367] distinguish between implicit (tacit, informal) and explicit routines. Process models are explicit routines.

2.5.2 User Participation in Workflow Articulation

Workflow management systems (WMS) provide active coordination support based on explicit process models. The trade press is full of workflow management success stories, claiming tremendous improvements in quality, costs, and service times [167, 532]. An overview of such studies [131] shows that several organisations experienced much shorter case processing times and improved efficiency. In one case this resulted in staff reductions, in another improved job satisfaction.

Rigid Workflow Management Systems

Severe limitations of WMS are also commonly pointed out [349, 350, 476]. One study of a printing company [69] shows that a number of opportunistic, ad-hoc behaviours allow workers to maintain a smooth flow of work. These include flexible prioritisation of jobs, utilising time while one job is being printed to prepare the next, processing urgent jobs before a confirmed order is received, emergent support of each others work, enabled by attentiveness towards the state of others' jobs, and allocating interruptible jobs to people who are working at the customer counter. When a workflow management system was introduced into this setting, the explicit goals included recording worker activity, supporting process management, centralised monitoring and enforcing quality standards. The system caused proceduralisation, individualisation and serialisation of work, disrupting the ad-hoc activities that had previously enabled a smooth flow. Entering data into the system created overhead, which was especially troublesome for short, urgent jobs. To remedy these problems, some shops started to use manual records during the workday and only used the WMS to record data at the end of the day. Bowers et al. [69] propose the terms '*workflow from within*' and '*workflow from without*' to distinguish the naturally occurring ad-hoc coordination from coordination automated by the WMS.

Another study observes similar workarounds in the optronics industry [212]. The process in this study is more distributed and multi-disciplinary than that of the printing shops. Initially, paper-based checklists were used successfully. When a computerised tool was built, a lot of data had to be filled in before an order could be passed on. Receivers of this information complained that much of what they got was incomplete and unreliable. Over-serialisation and overly strict enforcement of constraints are major factors contributing to these problems. To make the situation tolerable, social networks developed which facilitated informal coordination and conflict resolution. Hayes concludes, "*designers of workflow processes need to accept that uncertainty can never be eradicated through the automation of procedures*" [212].

Flexible Coordination Enables Process Improvement

A more successful application of automated coordination is described by Grinter [199]. In three software development organisations, structured coordination tools were perceived as helpful. The benefits included automation of dull tasks and increased awareness of the work of others. In one company the introduction of the system even led to unplanned empowerment of the software developers. Originally, a special committee controlled allocation of bug-fixing tasks to developers. When this committee became a bottleneck in the process, developers were authorised to assign tasks to themselves. Soon the developers started using the system to organise their own work, attaching priorities, estimates, notes, and links to related problems, improving knowledge management. Another study of software development observed that local innovations (e.g. compile and build scripts) were spread informally across the organisation [331]. The adoption and use of CASE (Computer-Aided Software Engineering) tools over time have also been observed to cause changes in policies, practices and organisation [386]. For instance, developers began to spend more time with users, and began to see the potential for systems spanning the boundaries of functional units.

Articulating Processes for Exception Handling

Twidale and Marty [499] developed support for packing of museum artefacts. They discovered that the majority of the data recorded was not about which artefacts were placed in which box. Instead between seven and ten times as much information was recorded about the *process* of packing, detailing who packed, re-opened or moved boxes at different times. These process data were instrumental in diagnosing and remedying errors. There were no a priori organisational procedures; the museum professionals were making the process up as they went along, as each artefact was unique, each situation a possible exception. In this continuous process improvement, the focus was on detecting and fixing errors, not on preventing errors through standardised procedures. Attempts at designing error prevention mechanisms did not work. Some of these mechanisms evolved with the practice and became used for error recovery instead [499].

2.5.3 Representations of Work

The problematic nature of process models is fundamental. Actions cannot be fully predefined; they can only be "*accounted for post hoc with reference to its intended effects*" [468]. Software designers should thus be cautious about trying to anticipate too much about any work situation [414]. The scope and use of *representations of work* has been a major issue within the CSCW community. This was the topic of the Suchman-Winograd debate [469, 537] and [29], as well as a special issue of Communications of the ACM [471]. The problem is not simply one of richer modelling notations, but more fundamentally of what can appropriately be captured in any model of a work process [31]. This view resonates well with our focus on the interactive use of models. Models may be useful for some actor in a particular context, not true or false. The relationship between a model and the reality that it represents is dialectic: Models do not just describe or abstract from reality, they reframe it, and shape they way we see the world [31, 445]. "*The meaning of such representations is not fully determined but may be investigated, elaborated, or revised through continued interaction with the actual work setting*" [296]. Sachs [427] summarises the differences between organisational views and practice-oriented views on process design (Table 1). It is thus not surprising that information systems seem to have led to a *dual reorganisation*: decentralisation of low-level decision making and centralisation of power and control [9].

Organisational view	Practice view
People produce human error	People discover problems and solve them
Deskilling is desirable	Skill development is desirable
Routine work and thinking is desirable	Development of knowledge, understanding, and deciphering is central to skills
Flexibility = interchangeable jobs	Flexibility = skilled people
Standard operating environments are necessary to the business	Collaboration and collaborative learning take place in communities
Social interaction is non-productive	Communities are funds of knowledge
Automation produces reliability	Skills and learning produce reliability

Table 1. Design implications of underlying assumptions [427].

2.5.4 Processes, Plans and Situated Actions

Project planning is a major part of process articulation. In cognitive sciences, a plan is defined as a sequence of actions designed to accomplish some preconceived goal. This paraphrases the definition of process discussed above, reflecting that process management deals with planned work. Suchman [468] notes that human actions are inherently situated and ad-hoc, and that plans must not be confused with action. Post-hoc, plans *account* for actions in a rational way. Pre-hoc, articulation of plans supports coordination and shared understanding. Plans are thus "*resources for situated action, but do not in any strong sense determine its course.*" "*The organisation of situated action is an emergent property of moment-by-moment interactions between actors, and between actors and the environments of their action. The emergent property of action means that it is not predetermined, but neither is it random.*" [468].

2.5.5 Process Models and Practice

The gap between models and the reality they reflect has also been observed elsewhere [143, 146]. Bandinelli et al. [28] give an overview of software process improvement studies. One of their main conclusions is that process agents experience a discrepancy between the quality manual and the actual process. This contributes to dysfunctional organisational learning systems, as actual practice is hidden from discourse. In other words, descriptive models of existing practice are crucial for process improvement [417]. The software process community has thus identified the need for harvesting local process models into a general format suitable for assessment and comparison. Feedback from practice into process improvement needs to be extended from de-contextual metrics to rich descriptive models of process history.

2.5.6 Process Diversity

Studies of software development processes have lead researchers to conclude that different methods are needed for different projects [314]. Williams et al. [534] note that simple, well-understood tasks like programming, should be distributed to individuals, while complex and uncertain tasks such as analysis and design require close collaboration. Cockburn [109] highlights *criticality* and *size* as the most important characteristics of processes, defining a 4×7 matrix of methodologies. He also notes that all projects are unique and that other characteristics are also important, insisting that the proposed 28 methodologies are just starting points for local adaptation. There is a trade-off between the number of dimensions in a process classification scheme and the amount of local adaptation needed. Cockburn [110] refers to other classifications with between 5 and 15 dimensions, and up to 37400 process types. Clearly it is infeasible to predefine methods for every one of these types. An infrastructure where local models from existing projects are harvested for reuse in similar future cases, is needed if we are to approach the complexity of this problem. Combination of elements and aspects from different methodologies is also required if the large number of different types is to be adequately supported (requirements R8 and R10). The complexity of these classification schemes further points to the need for flexibly combining different criteria when searching and browsing for relevant templates in a process library (R9).

2.5.7 Interactive Process Model Reuse

Although actual practice differs fundamentally from explicit descriptions, organisations rely on these descriptions to understand and improve their work [77]. Interactive models cannot be detached from practice, their purpose is to help people plan, coordinate, perform and reflect upon their work. Hence interactive models are not as much in danger of distorting or obscuring the intricacies of practice. Through reuse, interactive process models support knowledge creation and dissemination anchored in everyday work. The key challenge is how models developed for one situation can be applied in another setting. Technically, this can either be done directly by copying a previous model, or we can generalise models into *templates*. A template defines a starting point model and a modelling language adapted for a particular use.

Current applications of process modelling span several layers of abstraction, from theoretical lifecycle models, via organisational models of best practice, to the plans of actual projects. Layered reference models have been proposed both in the software pro-

process [241] and workflow literature [254, 528]. Aiming to support learning, our main concern is to understand the mechanisms that enable *integration* of models at different levels of abstraction. This is the objective of the process model lifecycle shown in Figure 8. In this framework, process models are divided into two categories:

- *Particular models*, which aim to support performance of work in one project.
- *General models*, which abstract common properties from a number of actual processes, represent normative standards for the organisation, or are templates for reuse and adaptation into particular models.

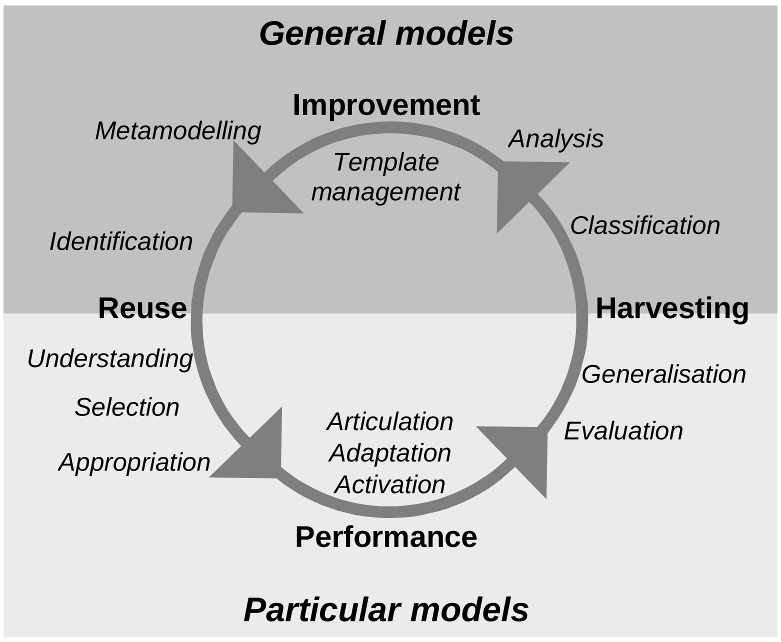


Figure 8. Lifecycle of process model evolution.

Applying a general process model to a particular situation is a case of *reuse*. Reuse may also refer to *copy and paste* of a previously developed particular model into a new process, i.e. reuse must not always occur via a general model. The process of generalising one or more particular models is called *harvesting*. The goal of harvesting is to provide templates that can be reused in the future, and to utilise practical experience as input to assessment and improvement of the general models. Following conventional terminology, the activity where people assess and update general models is called *process improvement* [28, 158, 355]. In process improvement and re-engineering, particular models are rarely used. For such initiatives to be cost-effective, they must target general models that are applied repetitively.

The activities of process performance, harvesting, improvement and reuse form a learning cycle. If one activity is not performed, the others suffer. This does not imply that all activities need to be explicit or encoded in software. A user may for instance manually improve a template based on lessons learned in a project, even without software support for generalising the particular project model. Similarly, a project model may be a passive plan, influencing practice although automated enactment support is not available. The following subsections discuss each of the four activities in turn.

Process Improvement

Although process improvement is outside the scope of this thesis, requirements from this area should influence the design of an integrated process support system. Process improvement and re-engineering has gained considerable interest as a methodology for organisational change and e.g. for creating more mature software engineering practices. A summary reports three directions of software process improvement [158]:

- *Standard processes* of best practice, e.g. ISO 9000,
- *Assessment* methods that evaluate the maturity of organisational processes, e.g. the Capability Maturity Model (CMM),
- Methods that utilise quantitative measurement, e.g. Quality Improvement Paradigm.

Standard processes are highly generalised. Assessment is usually performed through questionnaires with metrics concerning the efficiency and quality of each project [158]. Particular models are rarely available, so process improvement methodologies have focussed on the organisational level. Still, the need for local models is identified. For instance, to reach level 3 in the Capability Maturity Model, standard processes must be tailored to each project. The notion that best practice can be defined in general and applied universally, have caused a wide range of misfits with local cultures [460].

Legitimisation through Process Metrics

Organisational learning theories illuminate the role of process metrics. Argyris and Schön [19, 20] distinguish between single and double loop learning. In single loop learning, strategies for action are assessed when performance is not satisfactory. Double loop learning, which is needed for long-term improvement, requires that the underlying assumptions, norms and goals also be assessed. Metrics focus on efficiency or conformance with standards, directing attention towards single loop improvements. Underlying causes are obscured, inhibiting double loop learning. The many intangible differences between each particular process, implies that metrics should be treated pragmatically. Still, metrics has a role in learning and improvement, e.g. as a vehicle for *legitimisation*. Metrics quantify benefits and enable management buy-in. As discussed in section 2.4.4, it is important for social learning that a practice is legitimate and acknowledged by the organisation.

Process Reuse

Reuse involves *identification* and selection of suitable model fragments to reuse, *understanding* and learning about the effective use of these components, and *appropriation* of the component by the user community in the new context. The task of identifying a template suitable for your process is often cumbersome, so you end up building a new model from scratch, or using the same template for most of your processes. Identification of templates requires general software support like searching and navigation, but also structured template repositories (R9) and description of suitable usage context (R11). If several candidate templates have been identified, you must select the one most suitable for your particular project. Selection should be supported by qualitative and quantitative analysis (R8). The match between the current project and the suitable situation for each candidate template, is particularly important here [247].

Process Models as Boundary Objects

Reuse is a *learning process*. It is a vehicle for the establishment of successful teams and for individual competence building. Reuse may occur within and across communities of practice (cf. sections 2.1.3 and 2.4.2), hence interactive process models serve as *boundary objects* [462] between the subcultures of the organisation. You need to really understand a model, its assumptions, strengths and weaknesses in order for *knowledge* reuse, not just model copy, to take place. Among the techniques that can be included in a tool for understanding and appropriation are conversation and negotiation, rich process visualisations (R2), scenarios, model walkthrough, role play, analysis and simulation. Simple, user-oriented modelling languages (R1, R3) also facilitate comprehension.

Appropriation and Local Ownership

When a template has been selected, you have a starting point for the plan of your process. Now the project group must *appropriate* the model, adapt it to the local reality of their project, transform it from a general guideline to a locally used object. Local ownership of the models is essential for utilisation, adaptation and articulation of process knowledge to take place. This appropriation can be analysed as *perspective making* [60], the process of building and strengthening the common understanding of meaning, practice and identity within a team. Common understanding is essential for coordination and communication. A language can support perspective making if it is extensible (R12) and allows the representation of conflicting points of view (R13).

Reuse during Process Performance

During performance of the work, users need to add new details, remove irrelevant parts etc. of a model, possibly applying more templates as components in the main model (cf. Figure 9). Performance is the phase where the most intensive interplay of articulation and activation goes on, in the context of local models, owned, manipulated and utilised by the process participants. Tools and languages that support composition of models from parts, specialisation of models, inheritance, and integration of other data sources, facilitate this adaptation (R8, R10).

Process Model Harvesting

Local innovations are more readily available for reuse if they are generalised into templates. Although cases may exist where a particular model can be automatically converted into a template, the creation and refinement of templates often is a process of *re-engineering*. It involves manual adaptation of local models to make them suitable for different contexts. Typically, KM or methodology experts perform such knowledge capture. A template may be a deep structure, including both specific language constructs and an initial model. The system should include operations for generalisation and decontextualisation (R8), e.g. resetting attribute values to their initial state, removing details, replacing instances with classes [58, 241]. We should also be able to contrast and compare different particular models, highlighting common patterns, as well as differences, and enabling the combination of different particular models into one template.

- Extending the requirement for personalisation (R7), the system should allow people to put their personal touch on models. It should also identify the persons who have contributed to templates and those who have experience with using it, in order to facilitate social interaction (as discussed in section 2.4.4).

- Following the requirement to capture the process history (R5), *versioning and configuration management* should be supported [115]. The historical development of a process model as work progresses helps us to gain an understanding of how the model can function in another setting [90]. History enables humans to use ambiguity as a resource for learning [527].

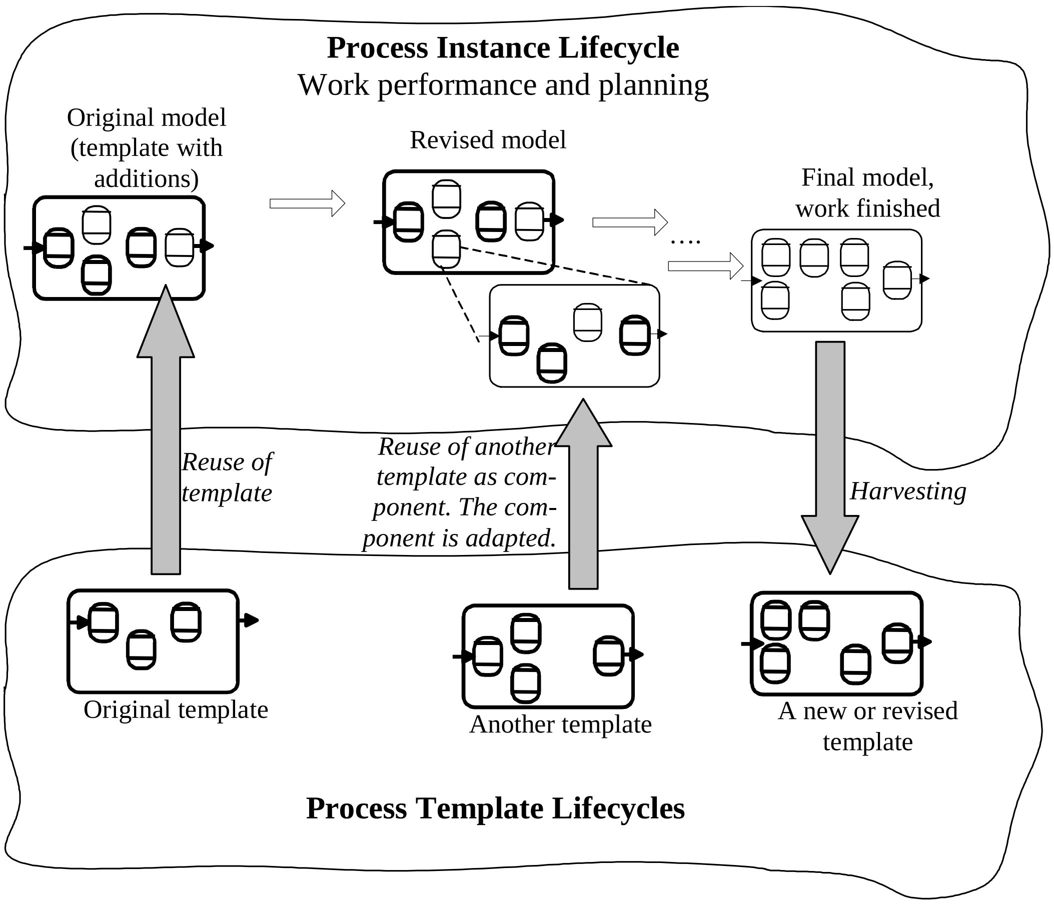

Process Instance Lifecycle
Work performance and planning
Original model (template with additions)
Revised model
Final model, work finished
....
Reuse of template
Reuse of another template as component. The component is adapted.
Harvesting
Original template
Another template
A new or revised template
Process Template Lifecycles

Figure 9. Reuse with process templates.

2.6 Summary

Table 2 summarises the requirements for interactive process articulation, activation and reuse. This list does obviously not capture every aspect of individual, social, organisational and situated needs that were discussed in this chapter. It pinpoints the major challenges at a level of detail suitable for assessing state-of-the-art and for evaluating the proposals of this thesis. It should be cautioned that this list is only one of many categorisation schemes that can be applied to this complex topic, and one should not reduce wicked, holistic problems to any such conceptualisation. A practical perspective also dictates that we should focus on the mutual interdependencies among these requirements, as reflected in the learning cycle of process model evolution (Figure 8).

Requirement	
R1	Simple language
R2	Visual, graphical language
R3	User- and domain-oriented language
R4	Evolving model semantics
R5	Local change, process history, versioning
R6	Incomplete models
R7	Personalisation and contextualisation
R8	Generalisation, analysis, metrics
R9	Structured template repository
R10	Composition of template fragments
R11	Description of suitable context of use
R12	Extensible language
R13	Multiple perspectives, interpretations and views

Table 2. Summary of requirements.

The requirements in this chapter focus on language and system features for interactive process models. They do not include requirements for process improvement or re-engineering detached from practice, and should thus not be considered a complete analysis framework for process modelling. A number of such frameworks already exist. Conradi and Jaccheri [115, 239] derive requirements for software process modelling languages. Their framework is a meta-process with six phases: elicitation, analysis, design, implementation, enaction and assessment. This chapter complements their list of requirements by looking in greater detail at the problem of integrating evolving models at different levels of abstraction. Rombach and Verlage [417] require process models that are natural, measurable, tailorable, formal, understandable, executable, flexible and traceable. The absence of input from organisational learning theories is common to these papers. Our previous work on the quality of process models [95, 96, 255] integrates theories about knowledge creation and social reality construction. The interplay between guiding (general, prescriptive) and tracing (particular, descriptive) process models has generated an underlying model similar to the one presented here [247].

Chapter 3
State of the Art

In this chapter, existing process support systems are evaluated according to the requirements presented in Chapter 2. This analysis aims to justify that

1. A new framework for flexible process support is needed.
2. Current process notations are too rigid and complex for most end users (section 3.1).
3. Conceptual modelling techniques can remedy some of these problems (section 3.2).
4. Current process support is too rigid for knowledge work (section 3.3).
5. A new combination of inheritance, composition, parameterisation, patterns, and other techniques is needed for process knowledge management (section 3.4).

Interactive models seem a suitable integrating scheme for currently fragmented research into ad-hoc, evolving and human-centred workflow management. Existing problems in workflow automation can be re-framed as interaction problems, and new challenges are identified. Section 3.5 discusses differences between interactive and adaptive workflow, and lists challenges for interactive workflow research.

3.1 Workflow and Process Modelling Languages

Models of work processes have long been utilised to learn about, guide and support practice. In software process improvement [28, 133], enterprise modelling [71, 168] and quality management, process models describe methods and standard working procedures. Simulation and quantitative analyses are also performed to improve efficiency [2, 295]. In process centric software engineering environments [14, 117] and workflow systems [532], model execution is automated. This wide range of applications is reflected in current notations, which emphasise different aspects of work. Carlsen [90] identifies five categories of process modelling languages: transformational, conversational, role-oriented, constraint-based, and systemic. The increased interest in modelling processes with UML [87, 169, 241, 315, 335, 435] requires that object-oriented process modelling also be discussed.

3.1.1 Transformational Process Modelling Languages

Most PMLs take a transformational (input-process-output) approach. Processes are divided into activities, which may be divided further into sub-activities. Each activity takes inputs, which it transforms to outputs. Input and output relations thus define the sequence of work. This perspective is chosen for the standards of the Workflow Management Coalition (WfMC) [530, 532], the Internet Engineering Task Force (IETF) [61, 475], and the Object Management Group (OMG) [378] as well as most commercial systems [1, 167]. IDEF [234], Data Flow Diagrams [174], Activity diagrams [381], Event-driven Process Chains [315, 443, 549], and Petri nets [553] are well-known transformational languages. In this section we explore languages within this paradigm, and assess their suitability for interactive systems.

Workflow Management Coalition and the Object Management Group

WfMC [529, 532] defines standards e.g. for process definition interchange [530] between systems. Processes are modelled with hierarchical decomposition, control flow structures for sequences, iteration, parallel (AND) and alternative (XOR) branching. Organisational roles and actors, tools and applications can be associated to activities. OMG defines workflow management as a common facility in the object management architecture [378, 440]. The aim of this architecture is to support runtime interoperability among business objects, components and legacy software. Its main focus is thus on exchange of data about ongoing process instances, not on modelling processes. OMG's workflow terminology is based on the WfMC (Figure 10).

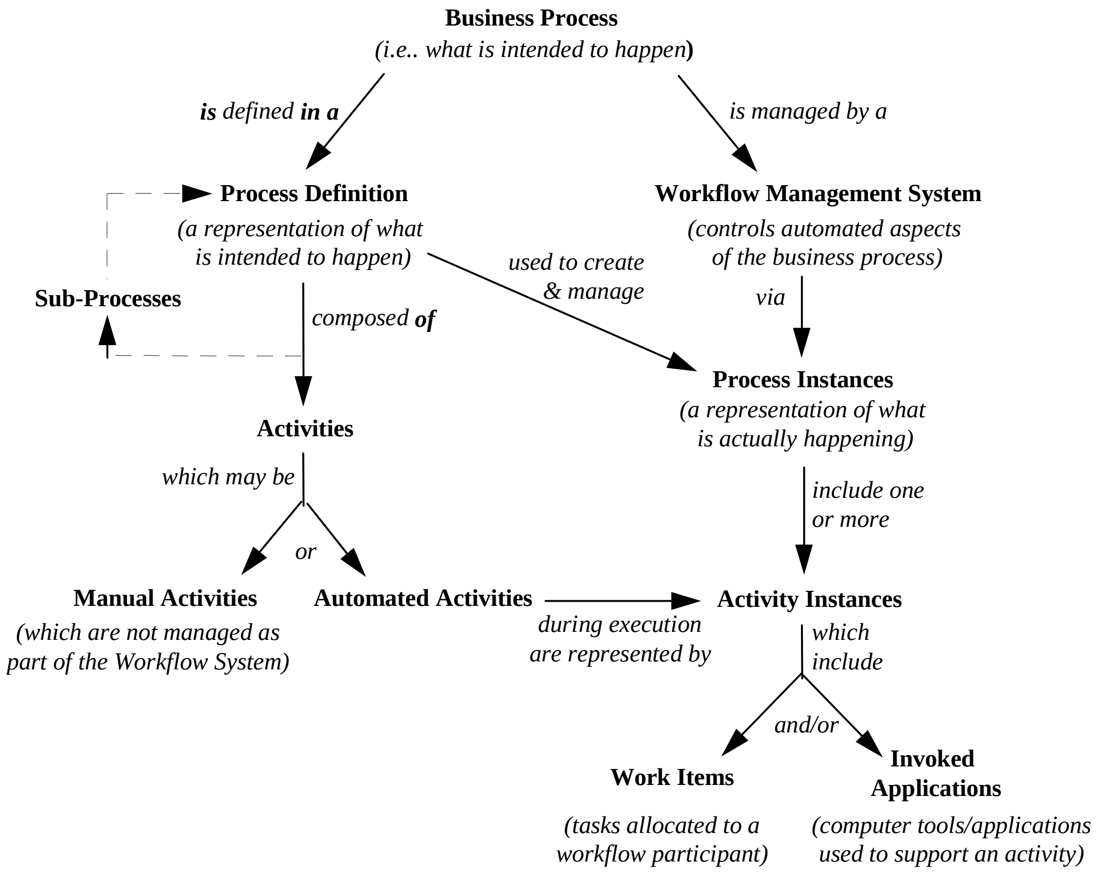

Business Process
(i.e.. what is intended to happen)
is defined in a
is managed by a
Process Definition
(a representation of what is intended to happen)
Workflow Management System
(controls automated aspects of the business process)
used to create & manage
Sub-Processes
composed of
via
Process Instances
(a representation of what is actually happening)
Activities
which may be
or
include one or more
Manual Activities
(which are not managed as part of the Workflow System)
Automated Activities
during execution are represented by
Activity Instances
which include
and/or
Work Items
(tasks allocated to a workflow participant)
Invoked Applications
(computer tools/applications used to support an activity)

Figure 10. WfMC core terminology [532].

BPML - Business Process Modelling Language

BPML [21, 459] defines a formal *web service* choreography interface and description language. Emphasising low-level execution, it contains several control flow primitives for loops (foreach, while, until), branching (manual choice or rule-based switch, join), decomposition (all, sequential, choice), instantiation (call, spawn), properties (assign), tools (action), exceptions (fault), and transactions (compensate). These constructs closely resemble block-structured programming. One strong point of the language is that it allows both manually controlled choice and rule-driven branching. However, as different primitives are used for the two cases, the automation border must be defined during process design. BPML has weak support for local change and unforeseen exceptions. Activities are not instantiated before they start to execute. They are thus not available for process overview or local planning.

Process Interchange Format (PIF)

PIF [304] was designed to standardise process definition among a number of research projects, including the Process Handbook [329], Virtual Design Teams [295], and TOVE Enterprise Modelling [168]. Although none of these projects primarily emphasised enactment, Bernstein [45] later proposed the use of PIF for flexible workflow management. PIF is a transformational language with constructs for modelling resources. Based on coordination theory [328], it defines a rich vocabulary for the relationships between activities and resources (*uses*, *creates*, *modifies*), and several inter-activity relations (*successor*, *prerequisite*, *cannot-be-concurrent*). PIF models also contain *decision* objects that select among alternative paths. The language does not separate among processes, activities and work items. It is thus simpler than the WfMC standards.

Petri Nets

Petri nets is a formal language for specifying dynamic behaviour. Petri nets are widely used in academic research on workflow management and software process environments [7, 156, 553]. Petri nets have a small set of basic constructs: places (circles), transitions (rectangles), and arcs. During execution, *tokens* occupy places in the model, representing the current state. Figure 11 shows an example Petri Net model. It demonstrates the basic expressiveness of control flow patterns: A waterfall *sequence*, *concurrency* (in specification), *synchronisation* (before design) and *choice* (C or Java).

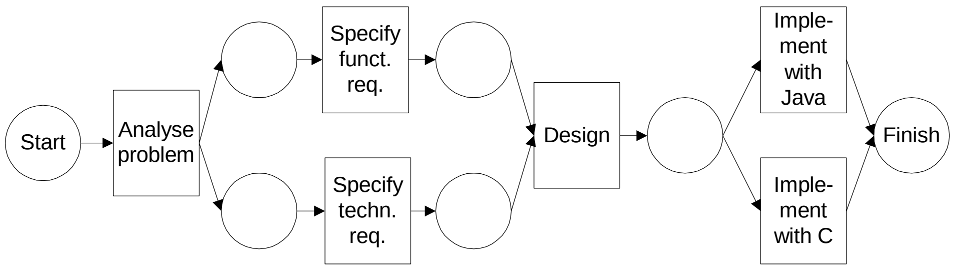


Figure 11. Petri net model fragment.

The transformational PML category has been subdivided into *task* and *state* oriented approaches [305], depending on which kind of element is represented as nodes in the flow graphs. Both of these approaches can utilise Petri nets. Most systems apply the state-oriented approach (tasks as transitions, as in Figure 11) [553], while UML activity diagrams are task-oriented (tasks as places) [63, 420]. The mapping of user level tasks and dependencies to formal constructs is thus not straightforward. While it may seem intuitive that tasks, as the active concept, should be represented by transitions, tokens spend their time in places during the performance of the tasks.

The main strengths of Petri nets are simplicity, formality and the large number of execution, simulation and analysis techniques that the research community has developed. Its most important limitations include theoretical, not user-oriented, constructs, and scope limited to process aspects. Lack of flexibility, abstraction (decomposition), and compositionality have also been highlighted [507, 533]. Hence user oriented visualisations and complementary modelling perspectives, e.g. for resources [154], object-orientation [297, 298, 354], and time, have been proposed [246].

Event-driven Process Chains (EPC)

EPC [434, 549] is a typical transformational process language applied in industrial systems. It is used in the ARIS modelling tool [433, 434] and the SAP ERP system [27, 443]. Extensions and variations of EPC have also been the subject of research [315], e.g. to handle other domains (products, resources) and more flexible enactment [443]. In EPC units of work are modelled as *functions*. Functions are enabled by pre-events, and cause post-events. Relationships between events and functions are modelled by arcs, AND, XOR, and OR connectors. As shown by van der Aalst [549], EPC models can be transformed into Petri nets. However, OR-connectors, representing a degree of uncertainty in the routing, are not well supported by the Petri net notation. Uncertainty gives rise to an exponential growth in the complexity of formal, operational models, because all combinations of alternatives must be explicitly represented.

Transformational Languages - Evaluation

Given the extensive use of transformational languages, most PML analyses focus on this category [115, 119, 195, 305]. The expressiveness of these languages typically includes decomposition, control and data flow, while organisational modelling and roles often are integrated. Aspects like timing and quantification [119], products and communication [115], or commitments [305] are better supported by other paradigms. User-orientedness is a major advantage of transformational languages. Partitioning the process into steps, match well the descriptions that people use elsewhere. Graphical input-process-output models are comprehensible given some training, but you can also build models by simply listing the tasks in plain text, or in a hierarchical work breakdown structure. Hence, the models can be quite simple, provided that incomplete ordering of steps is allowed.

3.1.2 Conversational Process Modelling

The language action perspective was brought into the workflow arena through the COORDINATOR prototype [538], later succeeded by the ACTION workflow system [339]. This perspective is informed by speech act theory, which extends the notion that people use language to describe the world with a focus on how people use language for coordinating action and negotiating commitments. The main strength of this approach is that it facilitates analysis of the communicative aspects of the process. It highlights that each process is an interaction between a customer and a performer, represented as a cycle with four phases: preparation, negotiation, performance and acceptance. The dual role constellation is a basis for work breakdown, e.g. the performer can delegate parts of the work to other people. Process models may thus spread out, as depicted in Figure 12.

This explicit representation of communication and negotiation, and especially the structuring of the conversation into predefined speech act steps, has also been criticised [85, 124, 271, 469, 472]. Minimal support for situated conversations, the danger that explication leads to increased external control of the work, and a simplistic one-to-one mapping between utterances and actions are among the weaknesses. On the other hand, it has been reported that the ACTION approach is useful when people act pragmatically and don't always follow the encoded rules of behaviour [124], i.e. when the communication models are interactively activated.

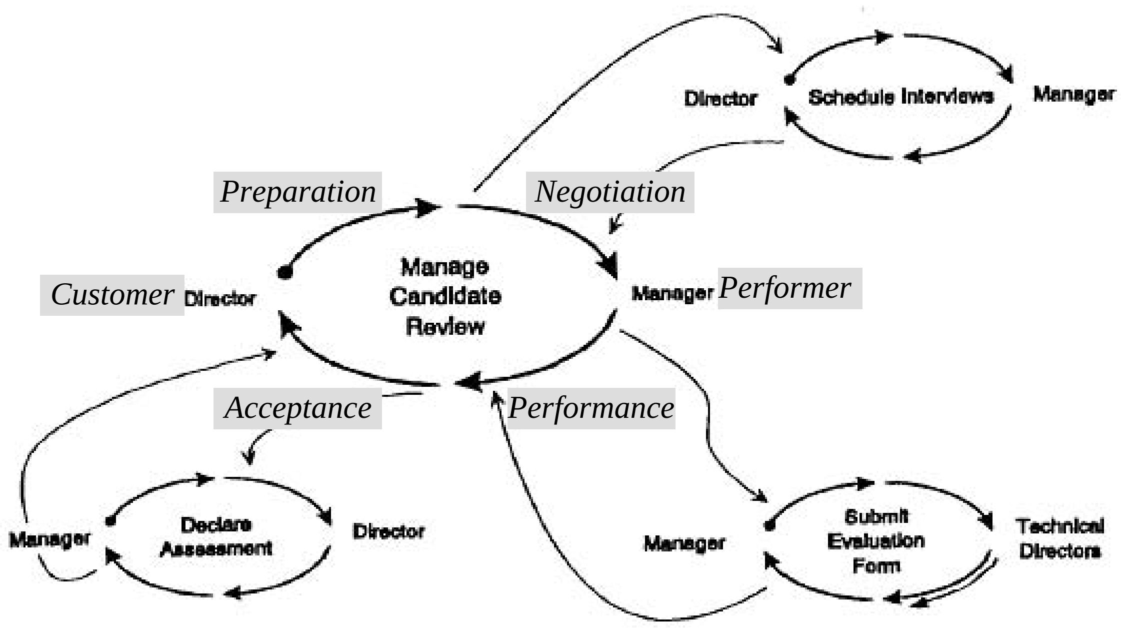


Figure 12. Action workflow model example [339].

E-Business Orchestration in ebXML

The language action approach has recently found new use in electronic business transactions and web services [270]. This has given rise to a range of standards dealing with process integration of web services across companies. ebXML [500, 501] aims to create a global electronic marketplace where enterprises of any size can meet and conduct business through the exchange of XML messages. ebXML defines a standard for business process definition that includes transactions, collaboration protocols, and simple negotiation patterns. Focusing on well-defined business domains, processes are standardised, and made available in public libraries to facilitate service discovery. Process evolution is not within the scope of ebXML, so for interactive models this standard is mainly relevant for defining simple, pluggable steps.

3.1.3 Decision Making Processes

Louridas and Loucopoulos [317] have defined a modelling language for representing the argumentation and rationale of a decision making process. Extending the vocabulary of Issue Based Information Systems (IBIS [113]), they break down the decision making process into goal (issue to be resolved), hypotheses (positions or solution alternatives), justifications (arguments pro and contra the hypotheses), and finally the decision that completes the loop. This is depicted in Figure 13.

Decision-making loops can be decomposed by spreading out new loops for sub-discussions, just like ACTION loops. The strength of this approach is that it combines an intuitive language for representing argumentation (IBIS) with a process loop. The decisions made and the rationale behind them are thus captured for later reference and learning. On the other hand, this notation does not fully cover the need for work process modelling, rather it must be combined with other approaches. The WINWIN prototype [67], extends design rationale and product models with negotiation around potentially conflicting goals. In RAPCEE [401] rationale constructs are integrated with process models. Here a *plan context* represents a strategy to fulfil a certain *intention* in given *situation*. Plan contexts are decomposed into executive contexts, choice contexts and other plan contexts. An executive context defines a process, which is automated by the system. A choice context requires human involvement in selecting among alternatives. Problems associated with explicating decision making rationale, e.g. the social barriers to participation discussed in Chapter 2, are common to these approaches. Although the languages reflect how people discuss rationally, experience indicates that many users do not feel comfortable applying the notation [188].

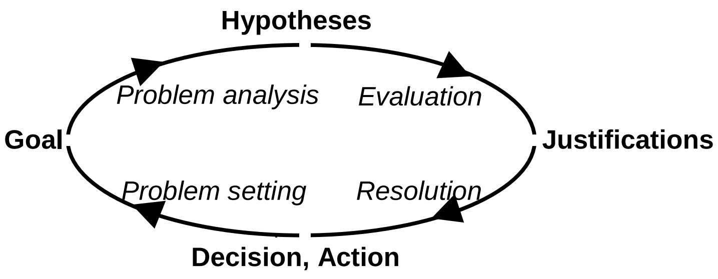


Figure 13. Decision making rationale and process loop [317].

3.1.4 Declarative and Constraint-Based Process Modelling

Declarative workflow approaches have also been promoted. Constraint based languages [142, 183] do not prescribe a course of events, rather they capture the boundaries within which the process must be performed, leaving the actors to control the internal details. Instead of telling people what to do, these systems warn about rule violations and enforce constraints. Thus, common problems with over-serialisation are avoided [183]. On the other hand, the resulting models are not very comprehensible. A graphic depiction is difficult since it would correspond to a visualisation of several possible solutions to the set of constraint equations constituting the model. The support for articulation of planned and ongoing tasks is limited. Consequently, constraints are often combined with transformational models [45, 142]. Constraints mainly capture outside control on the workflow, not articulation inside the process group.

Goal-Oriented Process Modelling

Agent-based process support environments often use goal-oriented, declarative process models [231, 303, 543]. Agents are assigned goals and constraint rules, but are left to work out for themselves the details of how to reach these goals. ALLIANCE [11] provides adaptivity, distribution and decentralised enactment for software-intensive processes. ALLIANCE represents the work breakdown structure (reflecting goals and sub-goals) in UML class diagrams, but does not define the ordering of the tasks. Scheduling of tasks is done dynamically during process enactment, influenced by modelled pre-conditions, inputs and outputs that reflect causal dependencies among tasks. ALLIANCE reflects the vagueness of explicit process models by capturing all knowledge about the current state as *fuzzy* statements. Fuzzy statements are assigned a probability between 0 and 1, going beyond binary logic. This opens up a richer set of alternatives to the agents that interpret and activate the model.

3.1.5 Roles and Their Interaction

Role-centric process modelling languages have been applied for workflow analysis and implementation. Role Interaction Nets (RIN) [457] and Role Activity Diagrams (RAD) [391] use roles as their main structuring concept. The activities performed by a role are grouped together in the diagram, either in swim-lanes (RIN), or inside boxes (RAD). The use of roles as a structuring concept, makes it very clear who is responsible for what. RAD has also been merged with speech acts for interaction between roles [40]. The role-based approach also has limitations, e.g. making it difficult to change the organisational distribution of work. It primarily targets analysis of administrative procedures, where formal roles are important.

3.1.6 System Dynamics

Holistic systems thinking [447] regards causal relations as mutual, circular and non-linear (cf. Appendix A.1.2), hence the straightforward sequences in transformational process models is seen as an idealisation that hides important facts. This perspective is also reflected in mathematical models of interaction [524]. System dynamics have been utilised for analysis of complex relationships in cooperative work arrangements [2]. A simple example is depicted in Figure 14. It shows one aspect of the interdependencies between design and implementation in a system development project. The more time you spend designing, the less time you have for coding and testing, hence you better get the design right the first time. This creates a positive feedback loop similar to "analysis paralysis" that must be balanced by some means, in our example iterative development.

System dynamic process models are used for analysis and simulation, but not for enactment. Most importantly, system dynamics shows the complex interdependencies that are so often ignored in conventional notations, illustrating the need for articulating more relations between tasks, beyond simple sequencing.

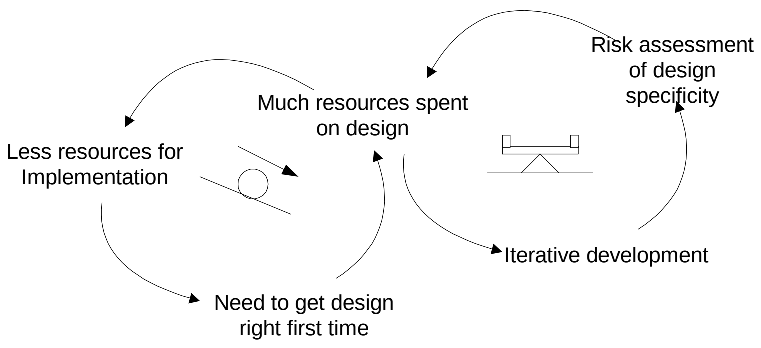

Risk assessment of design specificity
Much resources spent on design
Less resources for Implementation
Iterative development
Need to get design right first time

Figure 14. A system dynamic process model.

3.1.7 Object-Oriented Process Modelling

UML [381] has become the official and de facto standard for object-oriented analysis and design. Consequently, people also apply UML to model business processes. Object orientation offers a number of useful modelling techniques like encapsulation, polymorphism, subtyping and inheritance [315, 361]. UML integrates these capabilities with e.g. requirements capture in use case descriptions and behaviour modelling in state, activity and sequence diagrams. On the other hand, UML is designed for software developers, not for end users. A core challenge thus remains in mapping system-oriented UML constructs to user- and process-oriented concepts [226]. To this problem no general solution exists [315, 435, 467]. UML process languages utilise associations [238, 241], classes [229, 335], operations [87], use cases [104, 245], interaction sequence [229], or activity diagrams [104, 186, 381]. The lack of a standardised approach reflects the wide range of process modelling approaches in business and software engineering.

Modelling Processes in Class Diagrams

Class diagrams is a core language in the UML framework [381]. These diagrams represent the object classes and relationships (associations, aggregation, composition, inheritance) that form the static structure of a software system or application domain. Representing tasks and processes as classes thus enables the modelling of process structures with work breakdown (decomposition) and control flow dependencies (associations). Often, classes and associations are *stereotyped* as process model primitives [335]. A stereotype is an additional classification for a UML primitive, and may add specialised attributes, constraints and symbols.

Classes and Associations

Realising that the core knowledge represented in a process class diagram is captured by the associations between classes, rather than by the classes themselves, Jaccheri et al. [238, 241] focus on associations in their E^3 process modelling language. In E^3 *associations* are first class primitives that can be defined and specialised just like classes.

Classes and Operations

Carchiolo et al. [87] aim to support highly automated processes. They follow a programming approach to process modelling [389], where behaviour is articulated as *operations* in the process classes. An operation may include manual actions, and partial specification of methods is allowed. This scheme thus extends the conventional semantics of UML to include human involvement in execution, removing some of the rigidity that a fully UML-compliant process class diagram entails.

Use Cases

Use case models are often the first step in a UML project [244, 245]. Use cases capture the overall requirements on the system from the perspectives of various *actors*. When use cases represent activities in a process, the limited expressiveness of associations between use cases becomes a problem. While specialisation (*extends*) and composition (*uses*) can be represented, there is no way of articulating logical precedence relations (work and control flow). This makes use cases mostly suited for initial models, to be elaborated later with other languages [229, 435].

Activity Diagrams

UML activity diagrams [63, 420] are specialised state transition diagrams. Activities are distributed in swimlanes for different objects, similar to role-oriented PMLs. These models have expressiveness and syntax similar to Petri nets. Some researchers have found the fixed semantics of activity diagrams to be too rigid [435]. PROACTNET [104] is a software process engineering environment aimed at supporting human control, flexible scheduling, exception handling and incremental model definition. At the high level, it uses extended use case diagrams to capture loose collections of related activities. At the low level, specialised and extended activity diagrams capture resource needs, conditions, exceptions, inputs, outputs and products.

Integrating Approaches to UML Process Modelling

There are also a number of proposals for how different UML diagrams can be combined to model multiple process perspectives. DYNAMITE [435] follows this metaprocess:

1. Use case diagrams capture knowledge about current work practice.
2. The structure of tasks and resources are modelled in class diagrams, one class for each use case task. Associations capture control flow, data flow, and resource needs.
3. Concrete process specification class diagrams are elaborated. Task interfaces with input and output parameters are separated from internal task realisation.
4. A generic set of methods for changing state and transferring data parameters is defined for all tasks. State diagrams define the behaviour of task classes.
5. Event handlers that implement user level or model manipulation methods are assigned to task classes.
6. Based on the above models, an executable system is automatically generated.

Another integrating scheme is proposed by Hruby [229]. He uses class diagrams to represent structural aspects (organisations, business objects), use cases for dynamic aspects (processes and workflows), interaction diagrams for detailed dynamics, state or activity diagrams to model the process *lifecycle*, and also textual descriptions. Four models (structure, interaction, lifecycle and description) are made of both the processes and the objects, at five different levels (organisation, system, architecture, object, code). Each level refines and realises the specification on the level above. In total this methodology requires 28 different model views. Although the methodology is organised in repeating patterns, and uses the same techniques at different levels, the size is overwhelming. Researchers have thus criticised UML for separating rather than integrating different aspects (e.g. structure and process) [137].

Process Modelling in UML - Summary

The recent shift in OMG to focus on modelling (UML [381]), model driven architectures, meta-objects [377], business object frameworks and workflow management [332, 378] indicate an interest in model-driven process support also from the technical side. Whether these software-oriented approaches are transferable to interactive modelling, remains to be investigated thoroughly. Potential problems include UML's complexity, rigidity and system-oriented concepts [281, 137]. System-orientation also brings benefits, in that the conceptual gap between process modelling and software implementation is decreased. This is especially useful for workflow applications, where specialised solutions are built from standard components complemented with customised forms and business objects. In these applications, changes to the process model require recompilation of the application software. Software development processes, where the end users already know UML, is also especially suited for OO process modelling. As this section has shown, numerous mappings of process concepts to UML have been proposed. Integrating schemes combine several UML languages, but seem too complex for interactive work process modelling by end users. The lack of first class process modelling primitives is a major limitation of UML [137].

3.1.8 Other Explicit Process Representations

Numerous other textual, informal, or semi-formal process descriptions exist [119]. In project management, *temporal* considerations are important. This is evident in the frequent use of milestones and visualisations like Gantt diagrams. Ad-hoc *inscriptions on artefacts* also carry process information, e.g. for coordination and error recovery [499]. There has even been some research on utilising *programming languages* for process representation [115, 389, 541]. Process support systems are not the only area where *evolving, incomplete operational models* are needed. In tailorable systems, user interfaces, groupware protocols, method engineering, domain specific modelling, agent infrastructures, dynamic ontologies, multi-perspective and reflective systems, similar challenges are faced. The most important insights from these areas are discussed below.

3.2 Conceptual Modelling

Although suitable for articulating the aspects that users commonly want to share about their work, existing process languages still have a number of shortcomings with respect to the requirements uncovered in Chapter 2. Most notably, languages and models should be simpler, more domain-oriented, and easier to change. This section thus investigates general modelling techniques for solving these problems.

3.2.1 Instance Modelling and the Tyranny of Classes

Local modification of models is poorly supported in most languages. Instance modelling [5, 358, 393, 396] better represents particular situations (requirement R5). The complexity of having to take into account all present and future occurrences, has prevented dynamic change at the class level [213]. However, "*rather pragmatic changes to task instances are much more frequent than type evolution*" [368]. It is also easy to determine what a model element refers to, when it is a concrete individual (R1).

3.2.2 Property Modelling and the Tyranny of the Dominant Decomposition

The dynamic assignment of properties to object instances is an inherent feature of instance-oriented modelling languages [358, 396]. The *unique instantiation class condition* [24] limits flexibility of many frameworks. Often instances are not allowed to change class throughout their life span. State-dependent behaviour demands large classes that contain all features potentially needed by an instance during its lifecycle. The *dynamic objects* framework permits evolution of object structure and behaviour [24]. It is also possible to refer to past and future states of the system, and express semantic relations between time intervals. This allow us to capture the two-dimensional evolution of the modelled world and the users' understanding of it.

In order to accommodate multiple perspectives without increasing the complexity of models (R1), some have advocated an even stronger focus on properties as the core modelling construct. One such framework is developed by Opdahl and Sindre [385]. They argue that most modelling methods emphasise a few orientations at the expense of others, e.g. transformation, information, objects or actors. The problems of such approaches include limited freedom for the users to choose what to represent (violating R3), prescribed bias towards one methodology, reduced possibility of representing and integrating multiple perspectives (violating R13), and poor language extension capabilities (violating R4 and R12). To remedy these problems, objects are described through a number of *facets* [385]. The facets that represent a particular perspective are connected in a facet model. The language for a particular perspective thus becomes coherent and reusable. In aspect-oriented programming, *cross-cutting* allows a block of code (the aspect) to control behaviour of a wide range of objects [157, 351]. From a modelling perspective, this corresponds to declaring behaviour for all objects that possess a certain property. This is an interesting approach for integrating different viewpoints and software components around one model (R7, R13).

3.2.3 Encapsulation or Semantic Holism?

Senge [447] proposes *systems thinking* as a foundation for learning organisations. This paradigm replaces the linear causal relations of mechanistic thinking [357] with non-linear and circular dependencies [194]. It also replaces reductionism with *holism*: Every part in a system is related to every other to form a coherent whole. Though some researchers have experimented with system dynamics, most process modelling languages follow a reductionist approach. Their main building blocks are objects and binary relationships. Black box, closed system approaches seek to capture all dependencies between objects at the interface, hiding internal details [269]. While this approach is sensible for *constructive* design of the computerised parts of an information system, it is not self-evident that reductionism works equally well for requirements specification and interactive models [189, 263]. Systemic models can for instance better analyse the expected effects and interdependencies of alternative solutions. Conventional models are manageable, but the holistic perspective warns against approaches that assume, the interface between two subsystems to be completely specified. Models constructed with conventional languages, yet dominated by systems thinking perspectives, tend to be very complex and messy, as illustrated in Figure 15 and by [172].

For these models to be practical, we need multiple views that extract the features that are relevant for a particular task, role or situation (R2, R13). These views should be interactive, so that models can be edited through them. Views should also be customisable. Again a model-based approach is possible, i.e. to let interactive models control what to include in each view. Recently, people have experimented with integrating storytelling and rich models, by writing a story of what the model represents, and linking in model parts where they are relevant [292].

Holistic perspectives have also influenced information systems research. Kangassalo [263] distinguishes between *holistic*, *molecular* and *atomic* languages. With atomic semantics, the meaning of a model element is completely specified by that element alone, while semantic holism lets the meaning depend on the other elements in the model. Molecular semantics offers a compromise, where meaning is given by some parts of the model. This relates to the discussion above on how to design simple, user-friendly languages. A language with atomic semantics requires a lot of specific constructs, while holism allows meaning to be constructed by combining multiple elements, like words in a sentence. Property models similarly define what a model element means as a conjunction of all its properties. Holistic techniques are however poorly utilised in modelling languages, which emphasise formal, well-defined and static atomic semantics.

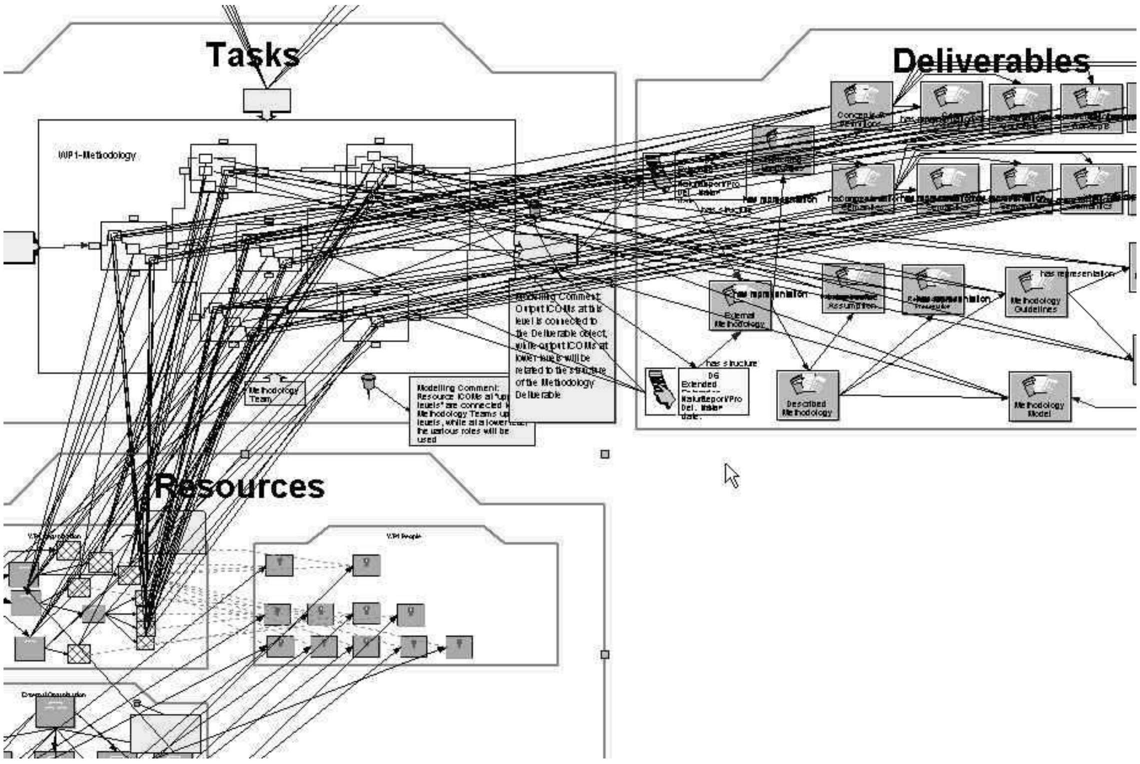


Figure 15: Snapshot of a reductionist enterprise model of a complex reality.

3.2.4 Articulating Vagueness, Incompleteness and Uncertainty

The SEEME modelling language contains elements for articulating the current state of specificity in a model [214, 215, 217]. It contains symbols denoting that additional knowledge is available but not modelled (or that it is articulated in another diagram), that some elements are hidden, that the model is known to be incomplete, or doubted to be complete, correct or appropriate. These symbols can be attached to any object or relationship in the diagrams, enabling integration with most visual languages. They are optional, and need not complicate the models. For an interactive model, incompleteness, vagueness and uncertainty are assumed characteristics. Whether the default interpretation of a workflow model fragment should be that it is completely or partially specified, depends on the context. Case studies are needed to investigate to what extent properties like uncertainty and vagueness can and should be explicated, but our requirements indicate a need for these mechanisms in articulation and negotiation of meaning.

3.2.5 Modelling Languages - Summary and Challenges

These two sections have provided an overview of languages for active process modelling. When faced with requirements for interactive process modelling, most current approaches share a number of weaknesses:

- Many languages are complex, containing numerous types and views not integrated in a systematic manner. This is especially the case for UML.
- In many cases mathematical, logical or technical concepts are applied instead of user or domain oriented. Petri nets and constraint based languages exemplify this.
- The languages that are precise and formal enough for automatic execution offer few opportunities for human contributions to interactive activation. The languages do not handle process models with varying degrees of specificity.
- The semantics of language elements is generally static and not easily adapted to local contexts or multiple perspectives.

There are generic modelling techniques that can help us meet these challenges. Instance and property modelling enable more flexible languages for local modification, multiple views and concept evolution. Semantic holism is a promising, but poorly developed, framework for designing simple and flexible languages.

3.3 Activation: Workflow Enactment and Beyond

Workflow is defined as "*the automation of a business process, in whole or part, during which documents information or tasks are passed from one participant to another for action according to a set of procedural rules*" [532]. This definition and most research focus on automatic activation. Miers [349] categorise organisations and workflow technologies according to degree of empowerment and autonomy, the dominant coordination mechanism, and the depth of relationships between groups (Figure 16).

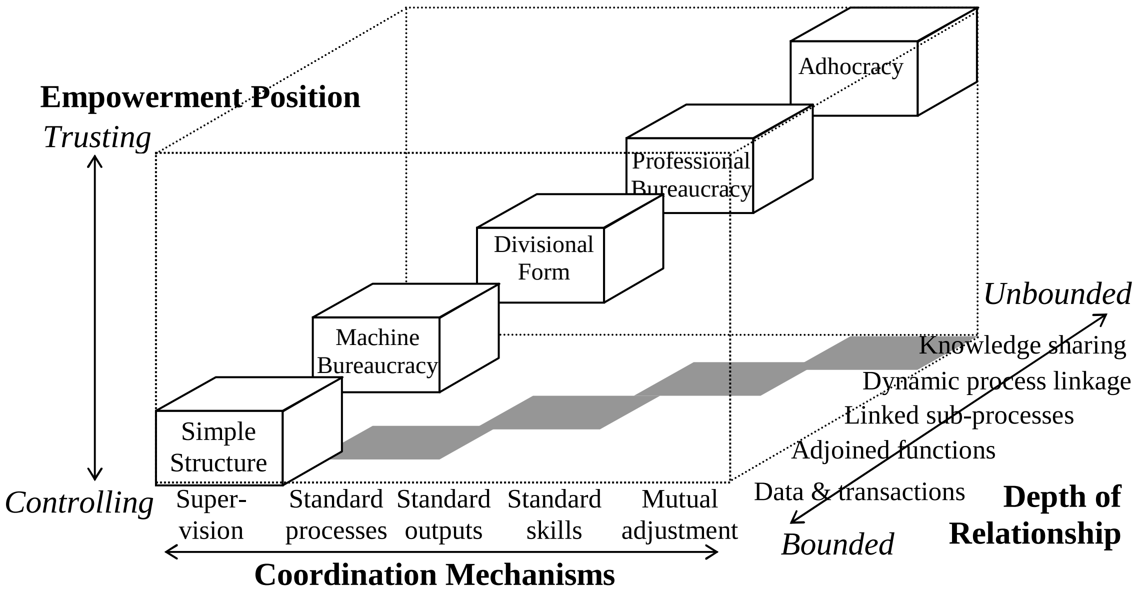


Figure 16. Miers' workware evaluation framework [349].

The top right corner is identified as the main challenge for further technology innovation. The interaction perspective attacks this challenge by viewing workflow as *"active support for planning, performance and coordination of work based on an evolving, more or less complete, explicit process model"*. This definition broadens the scope by

- Including groupware support for other forms of coordination (e.g. mutual adjustment), not just standardised processes and automated sequencing of tasks,

- Allowing incomplete and evolving process models, managed by empowered users,
- Integrating process articulation (planning), as part of the process, facilitating knowledge sharing and dynamic linking of process parts.

3.3.1 Static Workflow Management Systems

Static WMS [532] separate process definition (articulation) from process enactment (activation), and do not handle changes to the definition during enactment. These activities are supported by different tool components (Figure 17), and performed by different roles. Process definition is the work of process experts, while process participants perform work through workflow clients and invoked applications. Managers administer and monitor the work. This resembles the separation between conception and execution in Scientific Management [73, 485]. Production workflow follows mechanistic principles, as indicated by the fact that the enactment service is called the workflow *engine* [92, 357]. Static WMS aims to "*increase efficiency by concentrating on the routine aspects of work activities*" [177]. The core research challenges thus include transaction processing, interoperability, reliability, performance and scalability, monitoring and management of large numbers of routine, well-defined processes. These challenges are clearly different from the ones uncovered here for flexible and interactive workflow, so a detailed investigation of static WMS is of little relevance to this thesis.

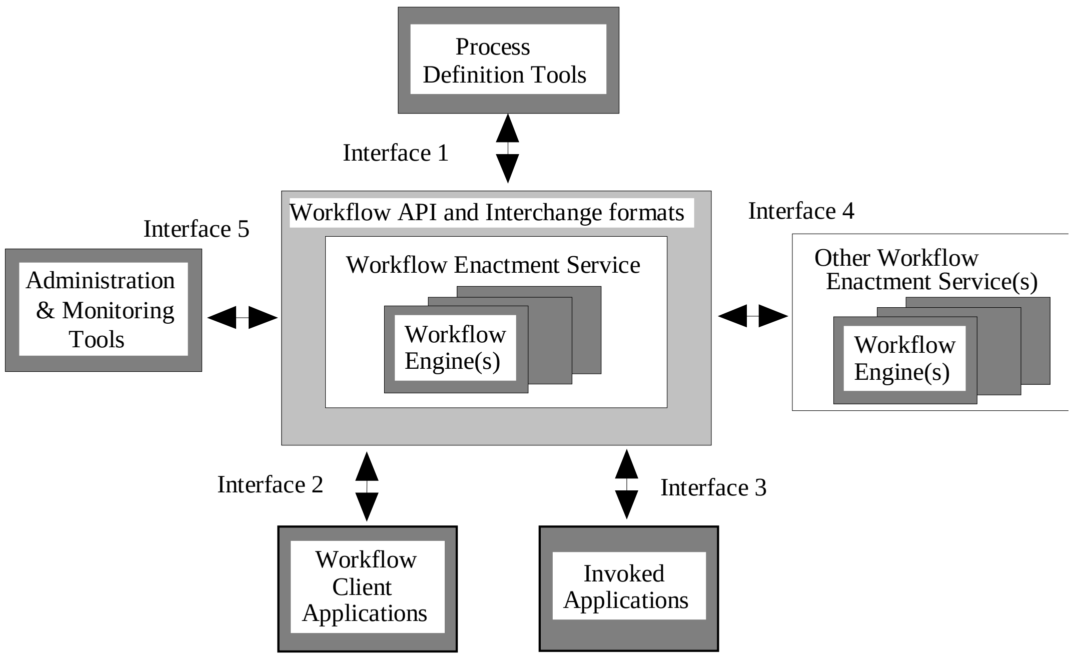

Process Definition Tools
Interface 1
Workflow API and Interchange formats
Interface 4
Interface 5
Administration & Monitoring Tools
Workflow Enactment Service
Workflow Engine(s)
Other Workflow Enactment Service(s)
Workflow Engine(s)
Interface 2
Interface 3
Workflow Client Applications
Invoked Applications

Figure 17. Workflow reference architecture [532].

3.3.2 Adaptive Workflow Management Systems

Flexible workflow is a hot research topic [46, 274, 276]. Most work within this area looks at how conventional systems can be extended, how static workflow systems can be made *adaptive* and *dynamic*. Research challenges for adaptive workflow include:

- Controlled handling of *exceptions* [103, 152, 156, 165, 275, 320],

- *Dynamic change*, migration of instances from an old class schema to a new [7, 98, 152, 153, 550].

Most research in this area recognises that change is a way of life in organisations, but still regards work as repetitive and prescribable. Within the community, an understanding seems to have emerged that change requires process definition and process enactment to be intertwined [152]. Most systems are however based on the premise that the enactment engine is solely responsible for interpreting the workflow model. In other words, users contribute by making alterations to the model, not by interpreting any part of it. Thus, the model must be formally complete to prevent ambiguity and deadlock from paralysing the process. This view of the engine as a Turing machine [523] complicates the models, because all process variants must be included [276]. It makes exception handling controllable, but cumbersome. Hence, some researchers challenge the feasibility of articulation by end users, arguing that they cannot be expected to change models correctly [213]. Although this research targets closed system automation rather than open interaction, many of its algorithms and design principles are relevant for interactive workflow as well. An overview of adaptive WMS also helps us get a clearer picture of what the innovative capabilities of interactive workflow is, and in which situations these capabilities are needed.

Workflow Enactment in Petri Nets

Several approaches to adaptive workflow are based on the manipulation of Petri nets [156, 553]. The execution of Petri nets is based on *tokens* moving along the arcs (flows) of the model. Tokens reside in the places of the model, and the token population (the *marking*) represents the current state of the process. Transitions are *enabled* when all of their input places contain tokens. An enabled transition may *fire*, causing one token to be removed from each input place and one token to be added to all of the output places. In this way, each token can be viewed as a thread in the work process. Threads may split and later merge at transitions.

Adaptive Enactment in Petri Nets

Exception handling is one area where the closed system assumption in adaptive workflow is temporarily relaxed, and human actors can influence the interpretation of the model. A lot of research into adaptive WMS has focussed on ways in which the token population in a Petri net can be altered in order to handle typical exceptions. The CHAUTAUQUA system [156] includes operators for splitting threads, and conflicts between threads are resolved interactively in a *merge* operation. Exceptions are handled by jumping to another activity. MILANO [6, 7] also allow user-controlled token jumps. Organisational policies determine which users are authorised to make what kinds of jumps. This technique can handle redo, loops, skipping an activity, parallelisation and sequentialisation. MILANO applies an integrated conversation tool to support negotiation in the exception handling process. These approaches handle simple changes to the execution order of tasks, partially meeting the requirement for local modification (R5). More far-reaching changes, e.g. re-definition of a process model, are poorly supported. Situated interpretation (interactive or manual activation) of incomplete models is not allowed, so the models must be formally complete and consistent. Users are supposed to react upon the arrival of tokens to tasks that they are responsible for, not to make proactive decisions regarding the flow of work.

The Dynamic Change Problem

In most adaptive WMS there is an implicit assumption that workflows are modelled at the class level (often called the schema), and that instances are enacted according to the rules specified for the class. In this context changes must be dynamically propagated from the class schema to ongoing instances [153]. Given the large number of instances that follow the same model, manual transfer is expensive. Dynamic change may also introduce deadlocks that halt some of the instances. Automated or semi-automated change management is thus needed. In Petri net models, the dynamic change problem is formalised as the search for a mapping of token markings from one model version to another. Ellis et al. [153] propose a combination of *immediate* and *delayed* transfer of instances to the new model. The model is partitioned into change regions. If a region is not changed in the new version, or if the changes applied in a region represent an up-sizing (the new version can do what the old one could and more), immediate transfer is possible for all instances currently in the region. Instances that reside in a down-sized or more complexly changed region, are not transferred until they leave the region (delayed transfer). This approach, called *synthetic cut over change*, is also implemented in MILANO [7]. While it ensures correctness, e.g. that no deadlocks are introduced in the migration process, the method is complex, and only handles a few change scenarios.

Metaprocess Support for Workflow Evolution

Ellis and Keddara [155] claims that "*a workflow change is a workflow*" in itself, arguing that metaprocess support for workflow change is needed. Neeb [366] makes a similar proposition, discussing to which extent metaprocesses can manage unforeseen and anticipated changes. In ML-DEWS (*Modelling Language to support Dynamic Evolution within Workflow Systems*) [155], change can be modelled, enacted, analysed, and monitored just like any primary process. ML-DEWS integrates the handling of instance and schema changes, and provide predefined change process schemes for typical approaches like *abort*, *defer*, *edit*, *continue* and *synthetic cut over*. Instance changes are handled as temporary schema changes. ML-DEWS is based on UML, with all workflow constructs represented as classes. Although their default change processes include an *ad-hoc* scheme, it is not clear (from [152, 155]) to what extent this case is facilitated by their implementation. The modelling language seems complex, rigid and system-oriented.

Casati et al. [98] also propose a language for specifying the change management metaprocess. A separate language is selected because it needs to be more flexible than the basic process definition language, and the performance and scalability requirements for change processes are not as strict as those for the core production processes. Casati et al. use Event-Condition-Action (ECA) rules for change management. An ECA rule declares what *action* should be performed when an *event* occurs and the *condition* statement is fulfilled. They are also used to control transitions in state diagrams [209]. The NETS-IN-NETS prototype [554] allows dynamic model interpretation through reconfiguration of the enactment rules at runtime. The enactment rules are represented as a Petri net metamodel. In other words metaprocess modelling and process modelling is done in the same language.

Exception Handling

In ADOME (*Advanced Object Modelling Environment*) [103], a WMS is built on top of an active object oriented database system. Other approaches [12, 97], have also built exception handling on top of transaction mechanisms in databases. ADOME was designed to minimise the need for human intervention in resolving exceptions. Consequently, trivial exceptions are handled by the system component where they arise. If the component is incapable of handling it, the exception may be forwarded to automatic exception handling. Exception handlers are defined declaratively with ECA rules or operationally with metaprocesses. If no automatic handler exists, users are involved, either interactively or with full manual control. If nothing works, the exception is regarded as a failure. ADOME advocate an ongoing harvesting of user-defined exception strategies into a repository for reuse. *Defeasible* workflow [319, 320] is a similar approach. Here workflows are modelled by justified event-condition-action (JECA) rules. The justification part describes the context in which the rule can be applied (R11).

This perspective on exceptions as a source for learning is not shared by Klein and Dellarocas [275], who define "*an exception to be any departure from a process that achieves the process goals completely and with maximum efficiency*". They use a taxonomy of coordination mechanisms in the PROCESS HANDBOOK [329] to classify and diagnose failure situations. Cugola [117] argues that inconsistencies in process models should be tolerated. This is partly implemented in the SENTINEL and PROSYT prototypes, but mainly for temporal deviations from an otherwise complete model.

Some designers of adaptive WMS thus suggest that designing exception handling as an interactive process, works better than full automation. However, human involvement is only triggered when no acceptable procedure is found. This approach applies interaction in exception handling, but not in normal activation. Consequently, incomplete and evolving models are not adequately supported. Human involvement is reactive, but not proactive.

MOBILE

The assumption that the workflow model is defined at the class level, underlies the design of the MOBILE workflow prototype [213, 228, 236, 366]. It distinguishes between

- *Flexibility by selection*, anticipated and modelled before enactment, and
- *Flexibility by adaptation*, modifying the workflow class definition.

Flexibility by adaptation can be implemented with or without dynamic change propagation to ongoing instances. Selection flexibility is achieved by *advance modelling*, where alternative paths are modelled before execution starts, or *late modelling*, where parts of the workflow class is left as a "black box" to be specified during execution. Interestingly, of the four evolution scenarios Heinl et al. [213] discuss, the only one being appropriately addressed by another solution than late modelling, is the *error* case, where the modellers have made a mistake. This indicates that interactive (late) modelling is more useful than conventional dynamic change and exception handling. The authors claim that although ad-hoc changes will occur, ordinary end users cannot be trusted to update models in an uncontrolled way. This claim is based on the view that workflow modelling is difficult, time-consuming and requires expert knowledge. Given the empirical evidence discussed in Chapter 2, the perceived impossibility of workflow articulation by end users seems to be a result of the class-based approach rather than an inherent feature of all workflow solutions. Instance modelling removes the great complexity of having to deal with all instances of a class in one model.

INCONCERT

INCONCERT is a commercial WMS that incorporates flexible collaboration features [1, 429, 430]. The original design goals included mutable process models, allowing for human judgement, and provision of a shared workspace for task performance. The system works on workflow instances, rather than classes. Templates with varying degree of structure are offered as starting points for local articulation by process participants. Instance-orientation also simplifies the modelling language. For instance, repetitive tasks generate new instances for each repetition, so loops are linearised. Although INCONCERT meets many of the requirements, it has some limitations. The system does not allow collaborative tasks with more than one actor. It does include some groupware features, like shared workspaces for documents, but not extensive support for e.g. negotiations or project management.

ENDEAVORS and TEAMWARE

TEAMWARE FLOW includes user-oriented process models as a design goal [478, 542]. Collaborative planning of work is approached through divide and conquer, with individuals taking responsibility of their own parts of the process. The focus is complete workflow models, not models consisting of more or less articulated fragments. Instance level modifications are supported, as long as they are in compliance with the class schema. ENDEAVORS [62, 222, 261] is the successor of TEAMWARE FLOW. It combines a number of techniques for exception detection, avoidance, tolerance and handling [261]:

- *Runtime dynamism*, late binding of resources, data, process structure and behaviour.
- *Configurable and partial execution*. The engine can be reconfigured to enforce different rules for each instance. Users can also define new rules (R4, R6).
- *Typed modelling*, extensible typing (R12), generalisation (R8) and specialisation of model elements.
- *Reflexivity*, changes to the model are allowed during enactment (R6).
- *Instances* can be promoted to templates, capturing local process improvement.
- *A library* of reusable (R9), composable (R10) model fragments.

Alongside INCONCERT, ENDEAVORS seems to be the WMS that offers the most support for the requirements uncovered here. It combines conventional adaptive workflow techniques with some interactive features, but support for local modification (R5) and incomplete models (R6) is limited.

Adaptive Workflow - Summary

Adaptive WMS offers important flexibility compared to static systems. Changes to workflow models affect running instances, and exception handling is supported. Adaptive workflow still shares the basic assumption of static WMS: that the process should be completely predefined at the class level. As we have seen, the situated and contingent nature of most project work demands local articulation in each project. New enactment concepts should thus be investigated.

3.3.3 Case Management Systems

Case management systems [350] such as VECTUS [348] and FLOW*er* [551], recognise the unique characteristics of each case. They facilitate manual handling of exceptions, manual selection and scheduling of tasks subject to predefined constraints etc. Cases are less complex than knowledge intensive projects and require fewer participants. Often the objective is to enable as few persons as possible to deal with the whole case, minimising hand-off overhead, speeding up the process and removing extra work in answering customer enquiries.

3.3.4 Towards Interactive Workflow Management

Workflow systems with less prescriptive models have been proposed as an alternative to the mainstream adaptive workflow research. Concepts like *human-centred* [117, 164], *mixed-initiative* [45], *evolving* [214], *emergent* [254], and *free* workflow [142] have been elucidated. These approaches typically include some of the following features:

- Soft constraint enforcement,
- Supporting proactive users in controlling the work themselves,
- Allowing incomplete, ambiguous models to be enacted, and
- Flexible combination of reusable model fragments.

This area of research is currently fragmented, lacking a common terminology and a well-defined set of problems [251].

Interaction as a Framework for Workflow Management

The interaction framework [523, 524] views computerised information systems as open. It directs research into new ways in which computerised and human actors can cooperate in order to solve problems. The notion of interaction machines (Figure 4 on page 15) can be further extended to *multi-stream distributed interaction machines* [523], which enable multiple users and software components to interact. Wegner and Goldin have looked at what interaction means for the expressiveness of models [524], establishing a domain and language-independent notion of interactive expressiveness and computability. Workflow systems can benefit from this approach,

- Because it enables more powerful exception and change handling, flexibly combining the capabilities of users and algorithms,
- Because multi-user, distributed groupware systems are better described as open interaction machines than as closed Turing machines,
- Because it provides a framework for reasoning about the use of models across different areas, and
- Because the expressiveness of interactively interpreted models better matches the contingencies of real work processes.

If an information system is to act as a knowledge mediator [22, 41, 93, 258, 425] between its users, it must be able to represent their knowledge accurately. Interactive systems are needed in order to mediate the non-determinism, vagueness and uncertainty of organisational realities [189]. A multi-stream interaction machine will appear non-deterministic to its users, due to *limited observability*. Any one user cannot know everything that occurs at all the interfaces to the system. Hence it appears non-deterministic. Correctness of open systems can never be proven in general, only for specific situations [524].

Interactive Workflow Enactment

A workflow engine activates a process model in order to support coordination. In most systems activation manifests itself as automated sequencing of tasks, though some also provide guidance based on modelled constraints [45]. The difference between automated and interactive enactment parallels that of Turing machines and interaction machines, as illustrated in Figure 18. In static and adaptive approaches enactment is completely controlled by the input workflow model. When change occurs, the model must be altered and reloaded. Interactive enactment accommodates change by intertwined user-controlled activation (re-interpretation) and model updates. Because user interaction is allowed the model need not be 100% complete and consistent. It may instead match the degree of specification currently found in user defined process models such as project plans.

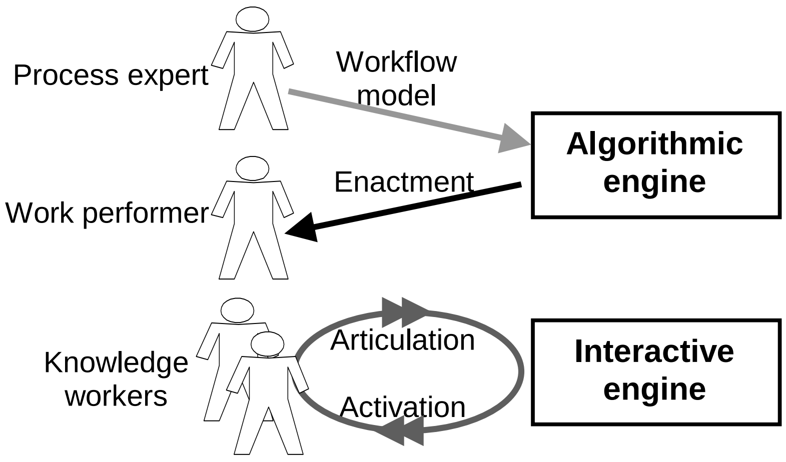


Figure 18. Algorithmic and interactive enactment.

Semantics of Models

The interaction framework includes a new look at how models are interpreted [524]:

- The Turing paradigm interprets models by *induction*, where every allowed behaviour must be modelled (or algorithmically deducible from the model).
- An interaction machine interprets models by *co-induction*, where everything is allowed that is not prohibited by the model.

Declarative languages follow the interaction paradigm in this respect [142, 183], but most operational approaches do not. This discussion is relevant for the ability of the system to support processes with varying degrees of specificity [45, 254]. Classic model interpretation assumes fully specified processes, hence the dynamic change problem. Interactive systems assume that models evolve to reflect improved understanding of the work. User involvement in activation is viewed as situated articulation, complementing or overriding predefined rules. This allows models with no structure, just an unordered list of tasks, and removes the common problem of system-required completeness that leads to over-serialisation of the flow of work [183].

Reconsidering Formality

The rigidity of current WMS architectures have been attributed both to the formal nature of the modelling languages and to the tight coupling of modelling and enactment. Agostini and DeMichelis [7] argue that neither conception is accurate, that simple models, easy to change for process participants, can remain formal and tightly coupled to enactment without causing rigidity. The real problem is not formality, but lack of interaction. Our perspective complements this view with the need for interaction also in model interpretation and enactment. This provides guidelines for avoiding the problems of formalisation while preserving the benefits. To enable automatic error checking, verification, deduction, simulation etc., the modelling language should have a formal definition with syntactic and semantic rules. This definition should however not prevent users from re-interpreting the model in situations that arise. Models should not be required to completely schedule all tasks, or always know which task is the next to be performed. The formal interpretation of the model is just one of many alternatives; it is the one that will be enacted if no further changes are made; it is the context-insensitive interpretation. What is needed is thus a language where complete models can be formalised but incomplete models are also allowed.

The interaction framework advocates *holistic enactment semantics*. It views the model as a system of autonomous components [524]. Each component can be formalised, but their interaction, controlled by users, cannot. Hence, the system exhibits emergent behaviour. This behaviour is non-deterministic and irreducible to algorithms, as shown by Wegner [523]. Thus, formality of components need not stop the system from mediating the situated, local, emergent, contingent, vague, and open nature of its environment [189]. This just requires interactive components and semantic holism. Most WMSs today apply atomic or limited molecular semantics, not holistic. In Petri nets the firing of a transition depends on the presence of tokens on its input places. Some molecular non-determinism can arise when overlapping token sets enables multiple transitions. This is often automatically resolved, or even prohibited as in *free-choice* Petri nets [549]. Some systems however let the users decide [535].

User Involvement in Enactment

A key question that the interaction paradigm puts to workflow designers is how users can be involved in the enactment process. As with other interactive systems, user involvement can be *reactive* or *proactive*. Reactive involvement means that the system asks a user to resolve ambiguity in the model. The follow-up questions in this scenario are: Which user(s)? When and how are they asked, and what support are they offered to solve the problem? Proactively, users can involve themselves by complementing or overriding the modelled scheduling of work. An example of this from the workflow literature is *opportunistic involvement*: That someone starts to work on a task although its inputs are not yet ready [142]. Such proactive actions can be facilitated by awareness notifications [415] and visualisations [253] of the current state of the process.

Shared Task Management

By not demanding that flow dependencies be included in the models, the interactive workflow approach on one extreme corresponds to *shared task management* [283]. In such a system no task interdependencies are articulated. Instead, information about tasks is available to all, and scheduling is left to the users. Experience has demonstrated that task management is useful on its own [283].

Linked Abstraction Workflows

Linked Abstraction Workflows (LAW) [164] implements an enactment framework that combines enforced scheduling of tasks with freedom for the end users. LAW accomplishes this by explicitly modelling the participants' authority to change the model. The authorisation scheme follows the work breakdown structure. For each task decomposition, these modification rights can be associated:

- CAN. The process is a recommendation, so the actors may use another method.
- STRICT. The specification is mandatory, and no changes are allowed.
- MUST. Specified steps must be performed, but new subtasks may be added.

This contribution is highly relevant for answering the question of which user to involve in interactive enactment. It provides clear rules for when human intervention can be local, and when it must be more centralised (pushed up in the work breakdown structure).

Guiding and Enforcing Constraints

As we discussed in section 3.1.4, constraint based reasoning has been applied to design flexible WMS [142, 183, 192]. A constraint rule should be fulfilled throughout its existence, unlike a transformational task, which is only active during the short time when it is executed. FREEFLOW components subscribe to notifications about events that occur in other components, e.g. due to a task dependency [142]. Constraint rules control the flow of work by determining whether a task can be completed or whether it must be performed again. In GPSG (Generalised Process Structure Grammar) [192] the process definition is open, so new constraint rules can be added dynamically during the execution of the process. Both document and task rules are supported. This interweaving of task and document structures enhance the expressiveness of the language and the richness of the coordination functionality.

Designing for Different Degrees of Specificity

Bernstein [45] articulates the need for WMSs that handle varying degrees of specificity in the process models, from unstructured to fully pre-specified. He shows how current tools either support very loose ad-hoc collaboration or automate fixed processes (cf. Figure 19). A case study is presented that highlights the need for integrating ad-hoc and predefined process fragments in one system. He proposes to combine coordination theory and soft constraints, enabling a flexible distribution of control between the system and its users. His design focuses on model activation, but does not facilitate process articulation by end users. No system prototype has been implemented.

Handling Vagueness and Uncertainty in Workflow

The SEEME modelling language was discussed in section 3.2.4. It allows users to explicitly state that they are uncertain about parts of a model, or that they have chosen not to explicate all details. This language has also been proposed as a starting point for "*evolving workflows by user-driven coordination*" [214]. The approach recognises that user involvement is needed in learning organisations, whose processes cannot be completely predefined. No workflow prototype has been built, but the SEEME language has been applied in a number of process modelling case studies [215, 216]. In these cases, the researchers learned that incomplete models are simpler and easier to understand for the process participants. For instance they discovered that the sequencing of activities should be minimised to only represent unavoidable causal relations. Hermann concludes that incremental structuration of a process by its participants is "*hardly supported by current groupware or workflow concepts*" [214].

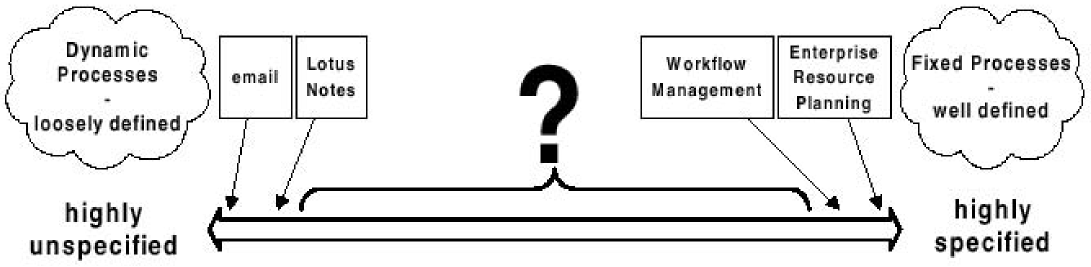

Dynamic Processes - loosely defined
email
Lotus Notes
?
Workflow Management
Enterprise Resource Planning
Fixed Processes - well defined
highly unspecified
highly specified

Figure 19. Different degrees of specificity in process models [45].

Extending the Scope of Model Driven Behaviour

Some WMS prototypes extend the scope of contextualised work support beyond workflow enactment. Several researchers have proposed to integrate groupware and workflow [18, 146]. MILANO [6] includes a multimedia conversation handler for negotiation and communication. XCHIPS [233] integrates a hypermedia collaboration support infrastructure and synchronously shared workspaces. ENDEAVORS [261] also includes a prototype integration with a communication tool.

Awareness servers notify users of a shared workspace about relevant events that have occurred [141, 405]. By keeping users up to date with the current state of the workspace, such a component supports coordination by mutual adjustment [352]. Integration of awareness and workflow has been proposed by a number of researchers [18, 415, 416, 428]. None of these proposals however have resulted in implementations or detailed investigation of the interplay between workflow and awareness engines.

INCONCERT (see above), INTERMEZZO [149] and some specialised systems [49, 68] have combined workflow with *access control*. Flexible configuration and reuse of access control policies is however poorly supported. Project management functionality has been prototyped in LAW [164], and INCONCERT is integrated with MICROSOFT PROJECT. The design proposed by Bernstein [45] (above) however goes farthest in combining model-activating components in a workflow architecture. This architecture is however not implemented, so the complexities that arise from integrating several components requiring specialised terminology (featuritis) is not dealt with.

3.3.5 Workflow Management Systems - Summary

This discussion has identified four main WMS categories: static, adaptive, interactive and case-oriented. Static and adaptive systems view work as repetitive and model processes at the class level. They differ in support for exception handling and whether changes to a class affect ongoing instances. Interactive and case-oriented systems view each process as unique, and facilitate articulation at the instance level. Case management systems deal with less complex processes, and the individual steps that each case consists of are often prescribed. Interactive WMS target project work, which is harder to structure in advance. Process learning is thus an integral part of the work. As we have seen, interactive workflow management is a research area in its early phases, and the most innovative proposals have not yet been implemented. The interaction framework illuminates key discussions concerning user involvement, formality and varying degrees of specificity.

Related Work

A number of different classifications of workflow management systems have been proposed in the literature. Georgakopoulos et al. [177] offer a comprehensive overview, refering to these alternative classifications:

- *Ad-hoc*, *administrative,* and *production*, distinguished by different degrees of human control, repetitiveness, and predictability of the supported processes. Production and ad-hoc roughly correspond to static and interactive in our framework, while administrative WMS resembles case processing.
- *Mail, document* and *process centric*, distinguished by different coordinative artefacts as primary elements in the workflow models. Abbot and Sarin [1] similarly distinguish between mail and database driven architectures.
- *Tightly* and *loosely* coupled. In a tightly coupled system the model is directly input into the workflow engine, while a loosely coupled workflow application is developed in a conventional software engineering project. Models are created in design and later programmed into the application.

Similar classifications are also surveyed by Carlsen [90]. Willumsen [535] outline approaches to execution of conceptual models in CASE tools during software development. The purpose of his approach is to validate the dynamic behaviour of the modelled system, and to enhance communication with the users through live representations (animation) and explanation of execution traces. Willumsen includes the user as an actor deciding what should happen next, pointing towards interactive models.

3.4 Mechanisms for Harvesting and Reuse

This section provides an overview of language constructs and operations that support process model harvesting and reuse. Like in the preceding sections, it is an emphasis on evolving, incomplete and ambiguous models that distinguishes this analysis from previous work [159, 406, 504, 540, 550]. The analysis is based on the organisational perspectives surveyed in Chapter 2. Tacit knowledge, gaps between models and practice, integration of knowledge management in everyday work, and the relations between trust, power and participation, provide core perspectives.

3.4.1 Generalisation of Process Models

The PKM reference model (Figure 8 on page 44) sees interaction between local and global process models as a foundation for knowledge management. This interplay requires that we understand what *generalisation* means with respect to process models. A model is a generalisation of another if it represents a larger number of cases. In object-oriented software, the meaning of superclass-subclass relationships is formally defined. What makes a *process* model a generalisation of another is not so straightforward. The main challenge is to generalise the dynamic and behavioural aspects.

In the PROCESS HANDBOOK [329] and contemporary workflow systems, generalisation may be seen as *adding* details to the models, increasing the number of possible paths through it [213, 540]. This approach assumes that the model contains every allowed behaviour. It has been called the *least-common-multiplier generalisation* [550]. Here models are treated as prescription for action, rather than as resources for situated adaptation [471]. Taken literally, this means that the root node in a specialisation hierarchy should include all the details of all other nodes in the tree. Such an approach will make general models hard to comprehend and difficult to maintain. Some of these problems are remedied by treating decomposition as a specialisation operation [329].

Models can also be viewed as socially constructed artefacts reflecting partially shared understanding of work, created and used with a rich background of tacit knowledge. From this perspective, adding details to a model means saying something more about the work it reflects; it means narrowing the scope of behaviour, i.e. specialisation. An empty model is viewed as a clean sheet, as full freedom, not as nothing-at-all. These *greatest-common-divisor* approaches [550], have local change (R5) of incomplete models (R6) as a foundation, enable structured repositories (R9) and simpler models (R1) that are easier to compare (R8) and manage configurations of (R5). The interaction framework thus advocate *co-inductive* model semantics in line with this approach (cf. section 3.3.4).

3.4.2 Classification

In section 3.2 instance modelling with dynamic properties was identified as a promising technique for simple, flexible and comprehensible models. Classification is still useful, helping actors to identify suitable templates in structured repositories (R9), allowing incremental model definition through inheritance (R10), and facilitating process analysis (R8). Classes improve cognitive economy (R1), and they are convenient in articulation. It is simpler to declare that "this object is a person", than to say that "this object should have the properties name, address, phone number and social security number".

Inherent classification, the requirement that instances must be defined by a class, is however problematic [396]. Classes are socially constructed; they are not essential structures in the real world. Parsons [396] shows that major problems in information management, including schema integration, evolution and interoperability, is a direct result of class-orientation, The solution to these problems can be greatly simplified with instance as a first order primitive. Multiple, dynamic classification allows different perspectives to coexist, be negotiated and represented within the same model (R13). Disagreements about classifications are more frequent than disagreements about the identity of individual objects.

Intensional Classification

Parsons [393] draw upon these insights to define a dynamic classification model, where objects belong to classes if and only if they posses all the defining properties of the class. Using the conceptual framework of Bunge [81], this classification method is called *intensional*. The CONCEPT D language [262] similarly utilise intensional containment relations to define new concepts through generalisation, specialisation, aggregation and value transformation.

Extensional Classification

The extensional perspective on classification views a class as a set of objects (the extension) with something in common [395]. Conventional subtyping is extensional, in that every object, which is in the extension of a subtype, also must be in the extension of the supertype. The extension of a class includes all actual and potential objects to which the class concept applies [81]. Extensional classification makes sense when you deal with instances, as it makes class membership (the relation between class and instance) explicit and straightforward (R1).

Prototype/Instance Frameworks

Not all modelling languages apply a class concept, some make do with just instances. Taivalsaari [483] discuss object or prototype inheritance as an alternative to class inheritance. Here, types are initially defined by their *exemplars* or *prototype* objects, i.e. by special instances rather than by class level constructs. New objects are created by *cloning* a prototype rather than by instantiating a class, and individual instances can be modified. The AMULET and GARNET user interface systems [358] are tailorable architectures that use prototype/instance inheritance. In these systems any instance can serve as a prototype for other instances, so any model element can be reused ad-hoc. Constraints are used to define rules for the inheritance of features from prototype to instance. Experience indicates that single layer models are easier to learn, more intuitive, more flexible, and less error-prone than two-layer (class and instance) models [358].

3.4.3 Inheritance

Inheritance is as a core feature of object orientation, combining reuse with specialisation. Specialised descendants inherit structure, interface, and/or behaviour from general ancestors. Inheritance thus simplifies models that are incrementally defined in a specialisation hierarchy. The detailed semantics of inheritance vary across languages. Taivalsaari [483] provides an overview of these many differences:

- *Single or multiple*, can a descendant have more than one parent?
- *Implementation reuse or interface subtyping*, the first sharing behaviour (code), the latter expressing specialisation. Subtyping assures automatic substitutability.
- *Class or instance* as primary object (as discussed above).
- *Dynamic or static*, is the descendant able to change ancestors during its lifetime?
- *Selective or full*, may descendants select which features should be inherited?
- *Acyclic* inheritance graph, or are cycles allowed?
- *Lookup scheme*, is the inheritance graph traversed top-down or bottom-up in order to find a feature? Is just the first definition used, or are all composed?
- *Creation copy or lifetime sharing*, do descendants just copy the features of their ancestors when they are created, or are there some way of sharing dynamic changes.
- *Deep or shallow copy*, are objects or just references to them copied?

The level of compatibility between a class and its subclass can be *strict* (same semantic behaviour), or non-strict at the levels of *signature* (full syntactic interface compatibility), *name* (operation signatures can be redefined), or *cancellation* (elements can be added or removed freely). Behaviour compatibility seems the most straightforward inheritance scheme for processes and other behavioural elements, but strict inheritance is of limited utility for evolving and complex systems [483]. Inheritance of process models is more complicated than inheritance in object oriented programming languages for these main reasons:

- Models consist of numerous objects, which are inherited together.
- Semantics (process behaviour) and not just syntax (blocks of code) is inherited.
- Models articulate human actions, possibly incompletely and incorrectly.
- Process models evolve during execution.

Numerous systems have applied inheritance to workflow and process models, trying to meet these challenges.

Inheritance of Operations

Conventional inheritance has been applied to process modelling. Carchiolo et al. [87] (cf. section 3.1.7) represent dynamic behaviour as operations, which are inherited from class to subclass. Operation *types* define the event types that are part of the operation (name compatibility), operation *templates* define the exact number and types of events (signature compatibility), while operation *defaults* also assign values to attributes (behaviour compatibility). For a given object class somewhere in the specialisation hierarchy, any operation may be defined as a type, template or default.

Inheritance of Associations

E^3 [238, 241] includes *associations* as first order primitives that can be defined and specialised just like object classes. Associations are also inherited from a class to its subclasses. E^3 models are structured into three levels. The language is defined as a hierarchy of classes and associations at the *creation* level. This level includes both kernel and user-defined classes, so the language is extensible (R12). At the *definition* level, general process models are defined as subclasses of the creation level classes. At the *instance* level, a particular model is generated according to the constraints of the template (following inherent classification).

Layered Policies

In OBLIGATIONS [58] an overhead transparency metaphor is utilised to combine multiple layers of process models. Each layer holds a portion of the model, and when the layers are put on top of each other (like transparencies), you get the total model. Similar to E^3, OBLIGATIONS has layers for general specification, local modifications and instance data. Each of the layers can be presented and modified alone. A layer may delete, create, replace and modify objects from more general layers (*cancellation* inheritance). A layer may include multiple templates, so multiple inheritance becomes a method for model composition (R10). The binding of instances to templates may go through a *surrogate*. Surrogates contain rules for which version of a template object to select, enabling late binding.

Petri Net Inheritance

Van der Aalst [550] investigate different forms of inheritance in Petri net process models, and list a number of transformations that can be used to generate subclasses:

- Adding a new alternative path,
- Adding a new parallel path,
- Adding new tasks as long as they don't alter the order of the old tasks.

Strict behaviour compatibility of allowed execution traces for each model is used as subtyping criteria. Two distinct forms of inheritance are proposed. With *protocol* inheritance, the subclass would have the same behaviour as its superclass if all the added tasks were blocked from use. With *projection* inheritance, the hiding of new tasks from the execution trace of the subclass results in the same trace as the superclass. *Lifecycle* inheritance is achieved if a combination of hiding and blocking can produce the same behaviour as the superclass. Lifecycle inheritance is found to be suitable for aggregating, comparing and structuring process variants [550], however the approach targets management of dynamic change and not organisational learning [541].

Requirements Met by Inheritance

Above four approaches to process model inheritance were described. Two started with classic OO mechanisms, while the others took transformational languages as their starting point, and derived inheritance mechanisms based on the dynamic behaviour of those models. All of these approaches enable generalisation and merging of local variants into a common schema for comparison and metrics (R8). Structured template databases (R9) can utilise their specialisation hierarchies. A two dimensional navigation structure like that of the PROCESS HANDBOOK [329] can be defined. (See Figure 20, where vertical decomposition and horizontal specialisation are combined. Positioned at a process element, you can move in four directions to explore related templates).

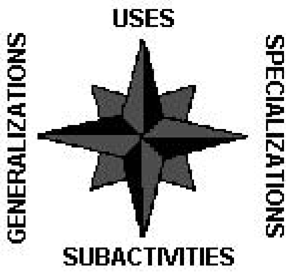


Figure 20. Process compass for navigating a template repository [329].

The languages of these systems vary from user-oriented (R3), simple (R1), graphical (R2) and extensible (R12) like E^3 and OBLIGATIONS, to programming-oriented. Petri nets have an advantage in that formal analysis methods exist (R8). With respect to local change (R5) and incomplete models (R6), OBLIGATIONS provides the most flexible solution. Both the OO approaches allow local change within predefined boundaries. The strict inheritance scheme proposed for Petri nets is rigid, and mostly directed at top-down control. E^3 and OBLIGATIONS support multiple views (R13), and separate instance data from local modifications, making it easy to reuse the specification part of any model, even that of ongoing processes (R8). Inheritance can however not help us meet the requirements most critical for social learning. The construction of a shared understanding among process participants should be better supported, by allowing representation, reasoning and reconciliation of different interpretations (R13). Other weaknesses include limited support for incomplete instance models (R6), composition (R10), and description of suitable contexts of use (R11). The following sections explore approaches to these challenges.

3.4.4 Process Model Composition

Composition of models from smaller building blocks is an essential reuse technique. In most projects, several templates must be combined in the work plan. This implies that harvesting should produce template fragments that we can easily put together in a number of ways. Alternative solutions to similar problems can also be combined into one template, encapsulating their differences into specific parts of the model. Model building involves both vertical and horizontal composition of fragments [313]. Vertical, or hierarchical, composition involves connecting a process model as a decomposition of a step in another model. Horizontal composition involves connecting fragments at the same level of detail.

Problems with Black-box Components

Encapsulation hides complex models inside a component with a clearly defined interface, so that it can be replaced by another model with the same interface. Encapsulation is a reductionist approach. The holistic, non-linear causal relations between activities imply that all dependencies cannot be captured by a simple interface [31, 71, 471]. Thus, there are pitfalls with encapsulation, e.g. that dependencies not captured by the interface are overlooked, remain tacit and are lost in subsequent situations where the components are reused.

Process Component Interfaces

The question is then what kinds of interface definitions are suitable for process models. In transformational models the interface of an activity consists of input and output *flows*. Some richer notations add *resource roles* [90]. Others define triggering *events*, pre- and post-*conditions* [115]. From a knowledge perspective, the *speech acts* that form the interface between a component and its environment, seem a suitable encapsulation mechanism [130]. Rather than just simple commands, the interface then allows negotiation across the boundary (R13).

Pluggable Actions for Vertical Composition

The Action Port Model (APM) [90] includes a novel approach to vertical composition called *pluggable actions*. In APM, units of work are modelled as *actions*. In addition to input and output flows, the interface of an action includes a resource signature, describing personnel, tools, information etc. needed to perform the action. Recurrent model patterns can be specified as *action clichés*. A cliché has one or more undefined steps, represented in its resource signature. In a particular process, local action definition are plugged into the cliché to take the place of the undefined steps. This enables local adaptation (R5) without changing the cliché. A similar mechanism is included in MOBILE [213], where *abstract variants* hide alternative sub-models from the main template (cliché), and *black boxes* represent parts of the model that will be specified later.

Runtime Components - Reuse through Brokering

Open Process Components (OPC) [175] manage both organisational and personal model fragments. A process component can start, suspend and abort execution, and tell other components about its capabilities and current state. This enables distributed coordination support. Relationships between components can be established dynamically through a *resource broker*, which matches the needs of a project with the capabilities reflected in personal process fragments. OPC even allows two components to have differing local semantics (R7, R12). Local change (R5) is facilitated by having components *delegate* their interpretation to representation objects, rather than inherit. The Internet workflow standard SWAP also support runtime components [61].

3.4.5 Projection of Multidimensional Process Models

Projection refers to taking a multi-dimensional model and removing some of its dimensions, e.g. removing resources or information flow from a process model. This is a main principle of reuse in the MOBILE environment [213, 236], where workflows are modelled from different perspectives:

- *Functional*, what is to be done, the tasks.
- *Behavioural*, when each task is to be performed.
- *Informational*, which data are used, how does it flow.
- *Operational*, how is the work done, the tools.
- *Organisational*, who performs the work.
- *Causal*, why the process is performed, its objectives.
- *Historical*, what happened during the performance.

Each perspective can be modelled independently of the others. This means that you can combine fragments from different perspectives to suit your particular process. The multidimensional approach enables simple models within each perspective (R1), guides composition (R10), and structures the template database (R9). Different models should be compared within each perspective (R8). This principle is applied in the PROCESS HANDBOOK [329], where related specialisations are grouped in *bundles*. The variants in a bundle typically differ with respect to one dimension only. For instance, a sales process may have one bundle of *operational* specialisations ("sell how"), and another for *causal* specialisations ("sell what"). For each bundle, a *trade-off table* captures the strengths and weaknesses of each variant. While this reductionist view on process models may seem too idealised, the perspective oriented approach points to important aspects of processes that one can isolate and represent as a template fragment. Removing a dimension from a model is a simple way of generalising it (R8).

3.4.6 Process Model Parameterisation

Dowson and Fernström [143] introduce the concept of *process variables*, which are bound to values during enactment, enabling tailoring of templates into particular models. Variables can represent products, roles, constraints, tools, goals and model fragments. Parameterisation resembles resource management in APM [90], where all of these aspects are part of the *resource signature* interface of each action, and resources can be allocated dynamically. APM's pluggable actions (described above), are similar to variable process fragments. From this perspective, specialisation involves constraining the range of values that can be bound to a variable, while generalisation eases constraints (R8). *Binding* of values can be predefined, rule-based or performed interactively by users, enabling local change (R5). A major strength of this approach is its open nature, that it can easily be combined with other techniques.

3.4.7 Process Patterns

Patterns of behaviour may be reflected in process models, and become evident when comparing several instances of similar processes. In software engineering and architecture [10, 130, 173], pattern languages have been developed for representing general solutions to recurring problems. Workflow patterns [552] capture routing schema, and show how they can be implemented in different languages. A great number of business process patterns have also been specified [412]. In management, *anti-patterns* describe troubling situations and how to avoid them [78]. Process pattern languages [130, 186, 467] add descriptions of the motivation, usage, range of application and underlying assumptions of each pattern (R11). This requirement is poorly met by current PMLs.

The NATURE project [201, 247] uses process modelling to describe methods and trace actions in early phases of software development. Their process models focus on *decisions* made in the process, in the *context* of a given *situation* to reach specified *goals* (R5). *Arguments* that support or object to the decision, are also captured (R13). The NATURE guidance mechanism is a pattern-matching engine that compares current product parts to situation descriptions of reusable process fragments in the repository. Telesius and Jaliniauskas [489] analyse a large number of business processes in government, banking and industry. Focusing on communication, they apply *common approximation patterns* to assess the reusability of workflow applications across these domains. This technique enables assessment of reusability across different communities of practice, bridging local dialects and frameworks of reference (R12).

3.4.8 Summary of Reuse Techniques

As this section has shown, no system currently meets all the requirements of integrated process knowledge management. *Inheritance* mechanisms for generalisation, structure and reuse must be complemented with *components* for knowledge combination and *patterns* that describe the environment where a model is useful. In addition to reusing model templates, the needs of knowledge intensive processes demand dynamic modelling languages, adaptable to the current vocabulary of each community (R12, R13). Therefore, metamodelling is an important feature. This demands a suitable underlying language model (metametamodel), capable of handling inconsistent vocabularies and deep concept structures, evolving in the process modelling language lifecycle.

3.5 From Active to Interactive Process Models - Research Challenges

This chapter has analysed flexible workflow management. It shows that a number of challenges remain, and that interaction is a key perspective for framing these problems. From an interactive, open systems perspective, we define workflow as

- *active* support for planning, performing, managing, and coordinating work,
- based on an explicit *process model*, evolving and possibly incomplete.

Conventional approaches [177, 532] separate process definition (articulation) from enactment (activation) [153], and can thus be regarded as active systems, but not as interactive. The separation between dynamic (adaptive) and static (production) workflow refers to whether changes to the definition of a workflow type can affect running instances [153]. Adaptive workflow emphasises automation and top-down control from classes to instances. Interactive workflow demands a bottom-up approach where the focus is on planning and performing local work process *instances*. Process definition (planning, articulation) is viewed as an activity that is part of the process it defines. Process templates are resources for adaptation rather than prescriptions of action. A clarification of the differences between these complementary approaches is needed. Table 3 presents a summary of these differences. The following subsection lists the main challenges for interactive workflow management systems.

	Static workflow	**Adaptive workflow**	**Interactive workflow**
Processes supported	Capital- and labour-intensive	Capital- and labour-intensive	Knowledge-intensive, creative
Process model	Complete, specified in advance	Complete, possibly with special constructs for late modelling of some parts	Incomplete, partially ordered and partially decomposed
Process instances	Several instances follow the same class schema	Several instances follow the same class schema	Instances are unique
WMS architecture	Separated definition, monitoring and enactment components	Most often separated definition, monitoring and enactment	Integrated planning, performance, management and coordination support
Modellers	Systems analysts and process experts	Systems analysts and process experts	End users (managers and workers). Experts facilitate knowledge management
Local changes during enactment	Not supported	Exception handling for changes like delegation and rescheduling	Changes are normal, handled at the instance level
General changes during enactment	Not supported	The dynamic change problem, for which many algorithms exists	Interactive dynamic change algorithms are simpler, rely more on human control
Activation	Automated enactment	Mostly automated. Some interaction at the model level (altering the model)	Interactive. Interaction at model and enactment level (manual re-interpretation)
Coordination	Automated sequencing of tasks	Automated sequencing of tasks	Automated sequencing and groupware services for mutual adjustment
Reuse	Through instantiation (fully automatic, but sometimes with initial parameterisation)	Through instantiation. Some copy and paste. Some allow ad-hoc addition of predefined tasks	Combinable templates available as resources for situated planning. Harvesting of local models into new templates
Research challenges	Transaction management, scalability, security, distribution, systems integration etc.	Dynamic change, exception handling	Articulation by end users, model reuse, enactment of incomplete models, extension of model-driven functionality

Table 3. Differences among static, adaptive and interactive WMS.

3.5.1 Articulation Support - Major Challenges

The limitations of current PMLs concern flexibility, simplicity and user-orientedness, all crucial for social learning through negotiation of meaning. Research into interactive process modelling languages should thus investigate:

1. How the basic premise that models are *incomplete*, partial representations of someone's understanding at a given time, can be better handled.
2. How can *language efficiency* be improved? How can we make simple representations whose meaning adapts to the current context, whose usefulness transcends specific situations without enforcing an idealised view?
3. How can the language facilitate *flexible* and ongoing articulation by end users? How can local modifications be straightforward?
4. How can domain and *user-oriented* concepts be utilised? What kind of reflection, extensibility and viewing mechanisms can enhance comprehensibility of the models, utilising the ongoing interaction between domain and model?

3.5.2 Activation Support - Major Challenges

Interactive activation extends the capabilities and adaptability of fully automated activation, which have dominated process support systems so far. Although a number of commercial systems and research prototypes involve users in exception handling, fully interactive activation semantics has not been implemented. These issues should be the subjects of further research in this direction:

1. How can we *involve users* in model interpretation? Which users should be asked to resolve which ambiguities in the models, and what support can they be offered?
2. How can the automated interpretation of process models be made more *contextual*?
3. Which *model-driven functionality* can we usefully integrate without complicating the models too much? Which services are most useful for different processes?
4. How can we make *customisation* of process support infrastructures simple, straightforward and user-friendly? How can we move from end-user programming to *end-user modelling*?

3.5.3 Process Model Reuse - Major Challenges

Section 3.4 gave an overview of the rich variety of mechanisms for process model reuse. While inheritance and composition dominates, patterns, parameterisation and projection also make significant contributions. Most of these approaches, with the exception of some pattern languages, focus on automating reuse. An interactive reuse framework could increase flexibility and usability by meeting these challenges:

1. How can *interaction* be utilised to simplify model reuse, making it more comprehensible to the users?
2. Can inherent classification be replaced by incremental, *need-driven classification* mechanisms, where templates and past models are classified based on the features of the current situation, rather than the long term management of the template library?
3. How can we *customise* the reuse mechanisms to fit organisational and user preferences as well as situated project needs?
4. What role can *metaprocesses* play in the reuse framework? Is it possible to offer active process support for reuse, adaptation, harvesting and improvement?

Chapter 4
Interactive and Emergent Workflow

This chapter introduces *emergent workflow* management [94, 254]. This approach utilises interactive modelling to meet the requirements outlined in Chapter 2. The term 'emergent' signifies that the process structure emerges from the work rather than being prescribed by outsiders. The core idea is thus to replace closed system automation with open system interaction as the fundamental design principle. Our prototype is called WORKWARE because it integrates workflow and groupware functionality.

The introductory section briefly describes the Action Port Modelling language (APM) [90, 91], which was our starting point. Section 4.2 outlines how APM has been refined to meet the requirements for process articulation by end users. The refined language is simpler, and it narrows the conceptual gap between planning and performance. Since the models are evolving and incomplete, the support environment is based on *interactive workflow enactment*. Section 4.3 introduces the enactment semantics of WORKWARE models, while the design of an interactive workflow architecture is outlined in section 4.4. After this overview, the second half of the chapter provides a formalisation of the design. UML class diagrams, state diagrams, and interaction sequence diagrams define the detailed activation semantics. This chapter thus describes how WORKWARE addresses articulation and activation, while Chapter 5 describes our solution to process model reuse. In Chapter 6 we show that the approach and techniques developed here are generalisable to other kinds of interactive models. The implementation of the WORKWARE prototype is described in Chapter 7.

4.1 Action Port Modelling Language (APM)

As the state of the art survey showed, there is a multitude of languages for work process modelling. The main problems addressed in this thesis do not require yet another modelling language. Instead continuity with existing approaches is sought. The APM language [90, 91] (Figure 21), is based on a conceptual modelling framework for information systems development, called PPP (Process, Phenomena, Ports) [206, 535]. The formal modelling framework of PPP includes execution and simulation [535], automatic generation of textual explanation from models [205], and specification of simplified model views [446]. PPP has also been extended with methods for performance analysis of information systems [384, 505] and organisations [72], and with versioning and configuration management in model repositories [15]. By basing our work on this foundation, we can thus integrate a rich set of useful techniques.

PPP process models are extended data flow diagrams [174], adding rigorous control flow semantics. APM simplifies the PPP flow interfaces (called *ports*, see Figure 21b), and offers a large set of constructs for modelling who performs the work (actors filling roles) as well as the resources (tools, materials, information) that they apply (Figure 21d).

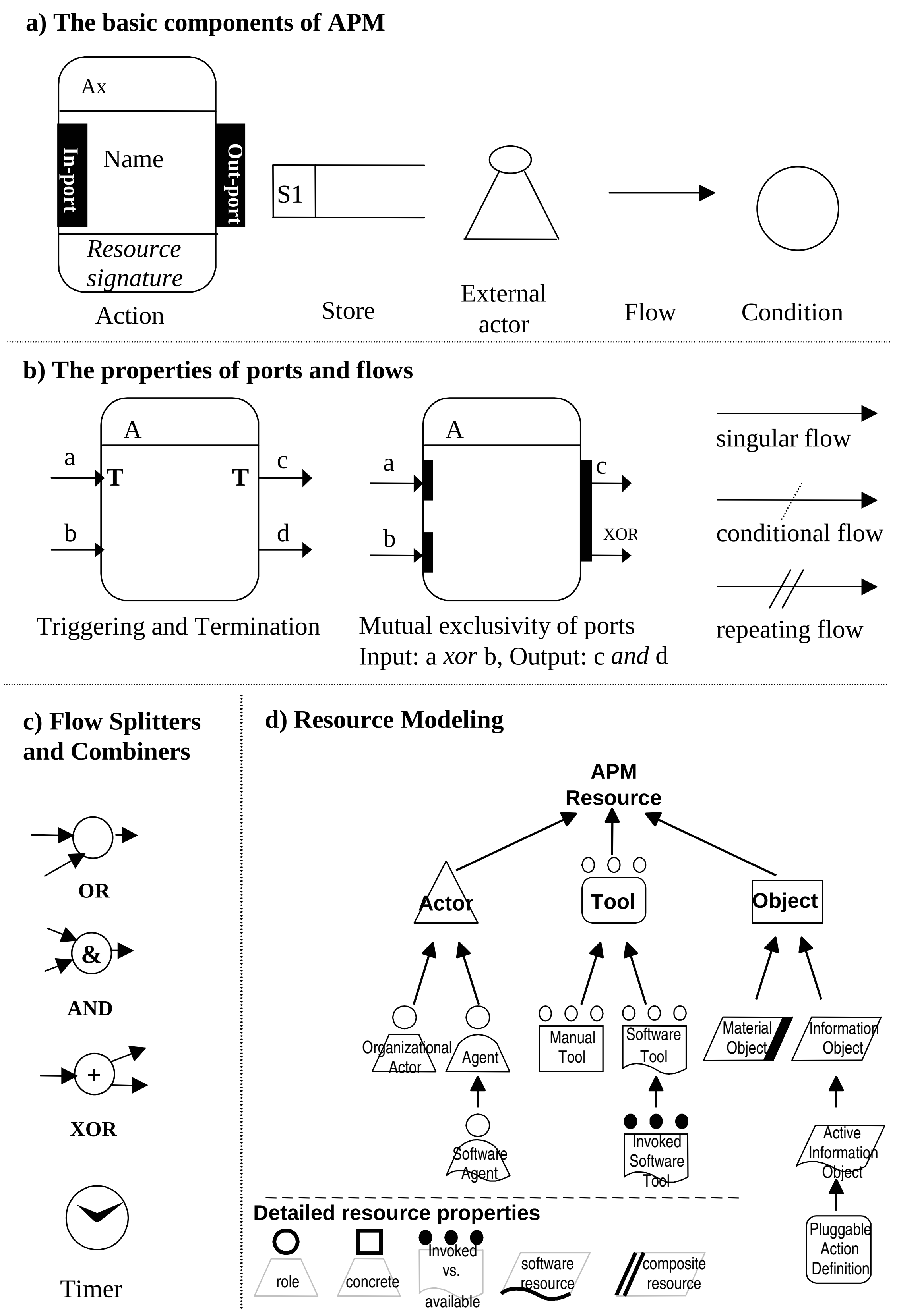


Figure 21. APM notation overview [91].

APM also includes innovative features for compositional reuse, and interaction patterns from the language action approach (cf. section 3.1.2). It is thus a transformational language with features from other paradigms (roles, objects, and conversations). APM processes are connected series of *actions*. Actions may be decomposed. Control flow signals that arrive at an input port trigger the execution of the action, while the flows attached to output ports may be triggered when the action is completed. Each port may be attached to any number of flows. All of the flows from one port are triggered together. Each action can have multiple input or output ports. Multiple ports signify alternative flow paths, hence for each instance only one of the input and one of the output ports will be used. APM primarily models control flow, not data flow. Data items and information objects are represented as resources, removing the spaghetti of data flows that make the models hard to comprehend.

The expressiveness of APM with respect to control flow patterns (sequence, choice, concurrency and synchronisation [552]) is similar to that of Petri nets [549, 553]. As in other transformational languages, work breakdown is the primary structure. Although decomposition is fundamental to most process modelling languages, some categories use different primary structures (e.g. roles or conversations). While role structures are important, work breakdown seems more fundamental, as project groups most often articulate what to do prior to deciding who should do it. The distribution of work among roles frequently changes without corresponding change to work breakdown, e.g. through delegation or re-assignment. Changes to the work breakdown, on the other hand, also entails changes to the work to be performed by some role/actor. The work breakdown structure thus seems more fundamental than the role structure.

As discussed in section 3.1, the aspects of work processes that conversational languages emphasise, depend more heavily on the political and cultural usage context, and the articulation of these aspects are thus more likely to meet resistance, increasing the gap between models and reality. Such languages do not seem the best starting point for facilitating end user articulation. Constraints and goal-based languages employ formal, textual notations, alienating many users. They are better suited for defining the constraints that the environment put on the process, than for supporting the process group in planning their own work.

Object-oriented schemes, e.g. various UML notations, also offer an integration of different modelling perspectives. However, the core concepts in UML are software-oriented. This demands extra effort in mapping user concepts to the software domain, as evident e.g. in the many different representation of "process" (e.g. as class, use case, activity, operation, or package). Continuity in methods and techniques is important to lower thresholds for adoption, and transformational languages are the most commonplace process modelling paradigm. Consequently, APM is a suitable starting point for emergent workflow modelling.

4.2 Interactive Modelling of Emergent Processes

Most languages used in workflow management and business process engineering are designed for analysts and experts who receive extensive training in modelling. For ordinary end users, we must simplify the language. To bridge the semantic gap between process articulation and enactment, we must make the mapping between modelling concepts and the users' environment (including IS and user interface entities) straightforward. While APM is a highly expressive language for business process modelling, it lacks interactive activation semantics and seems overly complex in some areas. Consequently, we decided to take a fresh perspective when adapting it to emergent workflow. The main contribution of this thesis is however not the changes made to the modelling language, but the activation semantics and the reuse framework.

4.2.1 Work Items

Avoiding unnecessary classification, the most basic concept in WORKWARE is the *workitem*, which correspond to actions in APM. A workitem represents a unit of work at any level of granularity or specificity. It may be decomposed into a process of sub-items. In the life-cycle of a workitem, such a work breakdown structure might be

- Added by actors planning their own work,
- Modified, capturing re-planning,
- Replaced by another process model,
- Removed, or
- Reported as performed, ignoring the advisory decomposition.

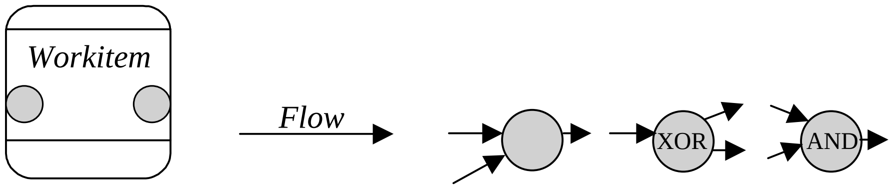
This means that whether a workitem is atomic or not, reflects its current state, and may change during its lifecycle. A system that does not include separate classes for atomic and composite workitems, can more easily accommodate this evolution. There is no need to change the class of the item alongside the instrumental change of adding or removing sub-items. The lack of a predefined, system-enforced classification of workitems into processes, tasks, activities, actions etc. thus makes the language simpler and more efficient. Users can express the same statements, but need not know about as many different language constructs.

4.2.2 Work Flows

Composite workitems are either *collections*, which have an unordered list of sub-items, or *workflows* where some sub-items are interconnected. Collections and workflows thus refer to models with different degree of structure. The interconnection network of a workflow is made up of two types of objects:

- *Flows*, representing dependencies between workitems, the sequence of work.
- *Decision connectors*, either joining or splitting a set of flows. We have three types of connectors, *Unspecified* (empty circle), *XOR*, and *AND*. The type refers to the logical relation between multiple input or output flows.

These three concepts, *workitem*, *flow*, and *decision*, constitute what is needed for enactment of workflows, which is the subject of the next section. A graphical notation for the workflow language is presented below. Each workitem has one input and one output decision connector, corresponding to starting and completing work on the item.



Workitems have resource signature, input and output decisions

Decision connectors. Both Join and Fork variants exist for all types.

Figure 22. The WORKWARE process modelling language.

4.2.3 Decisions

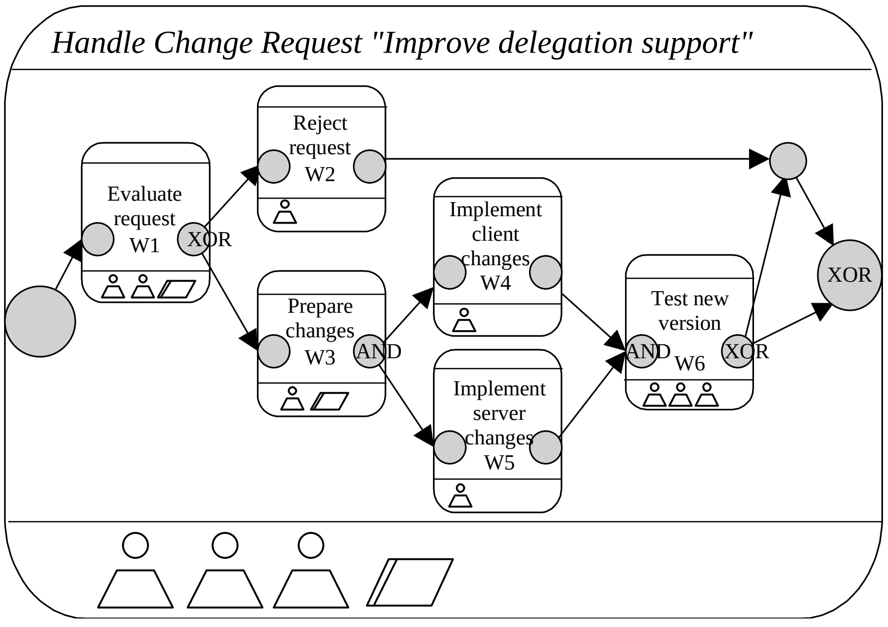
WORKWARE's modelling language follows APM in most aspects. The decision connectors represent an exception. They are a generalisation of the APM constructs *Port*, *Timer*, *Condition*, *Flow Combiner*, and *Flow Splitter* (Figure 21) [90, 91]. By generalising these constructs into a single class, we simplify the language without compromising on expressiveness. The rationale for unifying these constructs is that they all represent decisions regarding the flow of work. They fill similar roles in model activation, and need similar semantics. By unifying them into a common construct we thus also simplify the semantics. As we discuss below, the differences between the APM constructs can be derived from the relationships and properties of individual decisions in process models. For instance, decision connectors used as inputs and outputs in a workitem, correspond to APM ports.

4.2.4 Expressiveness

An example WORKWARE model is depicted below. It demonstrates, at least partly, the expressiveness of the language:

- *Sequence/Precedence*, W1 before W2 etc. (articulated by *flows*).
- *Choice*, in W1 we decide whether to go to W2 or W3 (XOR fork connector).
- *Parallelism*, W4 and W5 in parallel, either may start before the other (AND fork).
- *Synchronisation*, both W4 and W5 should finish before W6 starts (AND join).

More complex routing patterns [552] can be modelled by combinations of these elements and specialised decision connectors that will be introduced in section 4.6.



Handle Change Request "Improve delegation support"
Evaluate request W1
XOR
Reject request W2
Prepare changes W3
AND
Implement client changes W4
Implement server changes W5
Test new version W6
AND
XOR
XOR

Figure 23: Example WORKWARE model.

4.2.5 Resources

Workitems are performed by *actors* utilising other *resources*, typically tools and information objects. WORKWARE uses the APM resource taxonomy (Figure 21d). The actors, tools, objects etc. required for a workitem constitute its *resource signature*. The interface of a workitem consists of the input and output decision connectors and the resource signature. Properties distinguish among different kinds of resources. Objects in the resource signature of a workitem are initially resource *roles*. Roles must be *filled* by *concrete* resources when the item is performed. Concrete resources may be allocated to roles dynamically through integrated *resource broker* applications. For instance, we treat information as a shared resource. Flows may include information objects, but primarily reflect flow of work. Coordinating access to information is the duty of an *information resource broker* working in cooperation with the enactment engine. This solution helps us to simplify the process models by limiting superfluous data flows [90].

As shown in Figure 21d, properties are also used for distinguishing composite from atomic resources, mandatory (invoked) from optional, and software from material. The figure does thus not contain the complete hierarchy of APM resources. This framework can be used for modelling actual resources existing independently of workitems and for roles attached to workitems. Carlsen [90] shows how the actor constructs can be used to model organisational structures. However, when integrated within an enterprise modelling framework, the actual resources are typically modelled outside of the process domain, e.g. in employee databases or corporate directories. For WORKWARE we therefore added a relationship called *is-filled-by* between roles and concrete resources. Concrete resources in this case refer to any object that represents the actual resource. When a resource role is filled, it becomes concrete itself. This means that the resource symbol always tells users whether the role is filled or vacant. The is-filled-by relation can also be used between resource roles, facilitating indirect allocation. An example is shown in Figure 24 where the Project Manager on the main workitem fills roles in each of the three sub-items.

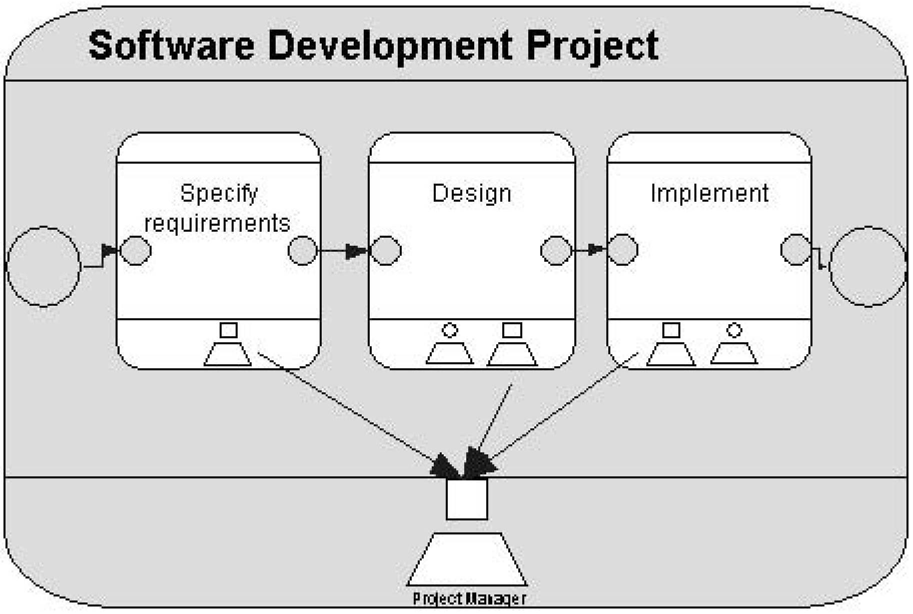


Figure 24. Simple model example with indirect personnel allocation through roles.

This use of the is-filled-by relation facilitates easy re-allocation, e.g. if the project manager is replaced by another person. It also supports the integrated modelling of work breakdown structures and role responsibility structures in reusable process templates. In APM, the is-filled-by relation was implicit. When two roles on an action and a sub-action had the same name, they were identical. The addition of an explicit is-filled-by relation for WORKWARE is caused by the shift in focus from process engineering to interactive work support. Usage experience also indicates that people often want to name roles differently to make them more meaningful in the context of each workitem.

The requirement that WORKWARE should support incomplete and evolving models (R6), implies that the system allows use of the generic resource classes in Figure 21d (Resource, Actor, Tool, Object). This is useful when modellers are uncertain of the specific characteristics of the resource, e.g. whether a meeting place will be real or virtual. Often the detailed characteristics of the resource are specified at the time of resource allocation, i.e. when one or more concrete resources fill the role.

4.2.6 Customer, Responsible and Participant Actors

Another WORKWARE extension to APM allows better-organised collaborative workitems. A collaborative workitem has more than one actor in its resource signature. Recognising the power of the ACTION customer/performer loop in supporting negotiation of incompletely specified work (cf. section 3.1.2), WORKWARE differentiates between these actor resource *role types*:

- *Responsible,* the main performer of the work, e.g. the project manager of a project.
- *Customer*, the actors ordering or paying for the result of the work, or an agent representing them.
- *Participant*, other actors involved in the team performing the work.

These concepts allows us to mimic the typical work breakdown structure of ACTION workflow, where the responsible performer of the main work item automatically becomes the customer for its sub-items. Below we will see how these role types are utilised by the components of the WORKWARE architecture.

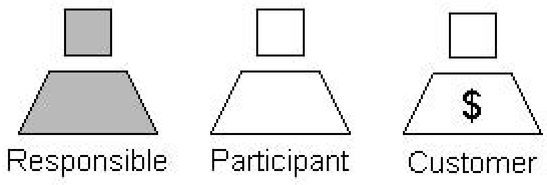


Figure 25. Symbols for actor roles with different role types.

APM, not primarily concerned with model activation, could leave concepts like role type implicit, allowing users to specify them through naming conventions, without extra modelling constructs. In interaction pattern clichés in APM [90], customer and performer roles were defined by naming.

4.3 Workflow Enactment as Interactive Activation

As we have seen, viewing the workflow engine as an *interaction machine*, generates new research challenges. We need to design an engine, which involves users in interpreting the model and bringing the workflow forward in the situations that arise. WORKWARE's engine is based on state transition models for workitems, decisions and flow elements. Some transitions are triggered automatically, while others are the result of events caused by users. The detailed activation semantics will be presented in section 4.5. WORKWARE models represent planned and ongoing, partially connected workitem *instances*. At any time, users may alter the model, changing the future interpretation of events by the enactment software.

4.3.1 User Involvement in Enactment

A key question that the interaction paradigm puts to workflow system designers is how users should be involved in the enactment process, both *reactively* and *proactively*. Situated interactive interpretation of an evolving, incomplete model requires that these challenges be solved:

- How is the responsibility for interpretation and action distributed between the system and its user in different scenarios?
- Which users should be allowed to, or responsible for, handling which situations?
- How can we separate mandatory from optional and advisory model fragments?
- When and how should users be asked to contribute, and how do we monitor that the process is not delayed unnecessarily?
- What support can users be offered to resolve enactment issues?
- How should conflicts between different users' interpretation and activation decisions be discovered and handled?
- How can all of these questions be addressed without complicating the modelling language, making it harder to comprehend and use?

4.3.2 Activating Flows

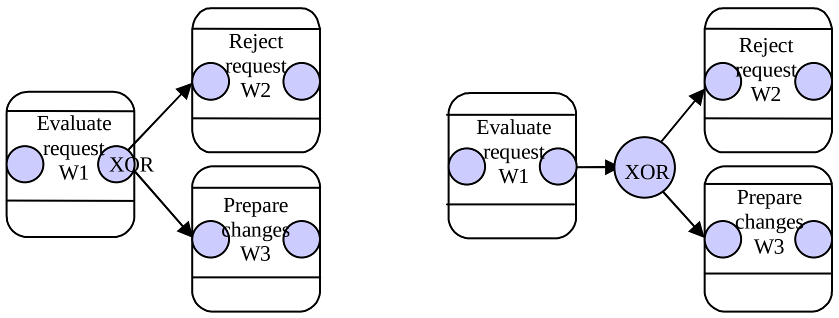
In WORKWARE's enactment semantics, work is progressed by *activating* flows and decision connectors. Activation may be triggered by users, or by the enactment engine interpreting effects of other events. Each flow represents *one instance* of a past or anticipated future signal. This view simplifies the handling of loops and repetition in the language. Every repetition involves a unique set of workitem, flow, and/or connector instances, i.e. we linearise the loops. This simple enactment model makes it easy to graphically depict the current status of a workflow, by marking all the elements that have been activated. We have applied a visualisation scheme where the workitems have different colour based on their states: red for not ready to start, yellow for ready, green for ongoing, grey for finished etc.

4.3.3 Decision Making

Decision connectors play an important role in the interactive enactment framework. A decision connector represents an issue to resolve regarding the flow of work. Some decisions may be straightforward to prescribe during the early planning of a process, while others must be made by human actors during work performance. Some connectors are easily interpreted by the enactment software, like an AND split or an XOR join, other decisions may not even be identified before they emerge as critical issues.

The default connector type is *Unspecified*, which doesn't say anything about the relationship between multiple inputs or outputs. Altering the type to AND or XOR adds constraints to the workflow, but needn't make the interpretation of what to do straightforward. What the engine does know, however, is that it should act when a connector is ready to be activated. If it is not able to determine which output(s) to activate, it must ask a user what to do next. We call this the *fallback* mechanism.

The position of a decision connector influences the interpretation of the model. If manual decision making is required, the engine asks the actor responsible for the workitem that surrounds the decision. Input and output decisions are thus left to the person responsible for the local workitem, while other decisions are to be made by the one responsible for the parent workitem. In the model in Figure 26a, the decision whether to prepare or reject changes is to be made locally in *W1*, while in Figure 26b the decision is part of the parent process.



a) Local decision in W1. *b) Higher-level decision.*

Figure 26. Modelling decision-making authority and responsibility.

This constitutes a simple and powerful mechanism for assigning responsibility and authority for decision making during the integrated planning and performance of a work process. The term 'decision' signifies that these objects capture decisions regarding the flow and scheduling of work. For a workitem, the input connector captures the decision to start the work, and the output reflects the completion decision. This is a unifying mechanism for interactive enactment, since both automated and manual decision making is supported. In addition to reactive fallback, proactive users can make unscheduled decisions without having to resort to modelling, e.g. when faced with exceptions. Over the whole range of activation modes, there is thus no need to take the process "out of the system", a well-known problem in the cases surveyed in Chapter 2. Every decision can be captured in the workflow trace. Which activation mode to apply, is handled in the situations that arise. Consequently, WORKWARE can handle any degree of specificity in the model, from unstructured (no flows, decisions left completely to the users) to completely predefined (all decisions automated).

The usefulness of automated enactment increases with the degree of specificity in the models. When flows connect workitems, the engine reasons about workitem state changes, automatically activating flows and enabling the next items. If a model contains no flows, just a workitem collection, the engine offers little active support. In this case, the users have full control of the process, and they have to do all the work of updating workitem states themselves. Since all model elements are instances, there is no conceptual gap between process definition and enactment. Unlike conventional systems (Figure 17 on page 62), the model editor and the work performance tools provide complementary interfaces to one, integrated model of concrete work.

4.4 Conceptual Design of an Interactive WMS

So far we have seen that interactive models can guide the design of workflow languages and enactment semantics. We will now look at how interaction gives a fresh perspective on system architectures, bridging the gap between articulation and activation. An interactive WMS architecture is depicted in Figure 27. It has three layers:

- A repository containing the *workflow models*, which are articulated and activated in the system.
- A number of *model interactors* that utilise the models to provide contextual functionality. A typical interactor integrates activation and articulation services.
- An *integrated user interface* that enables personalisation in addition to the contextualisation provided by the workflow models.

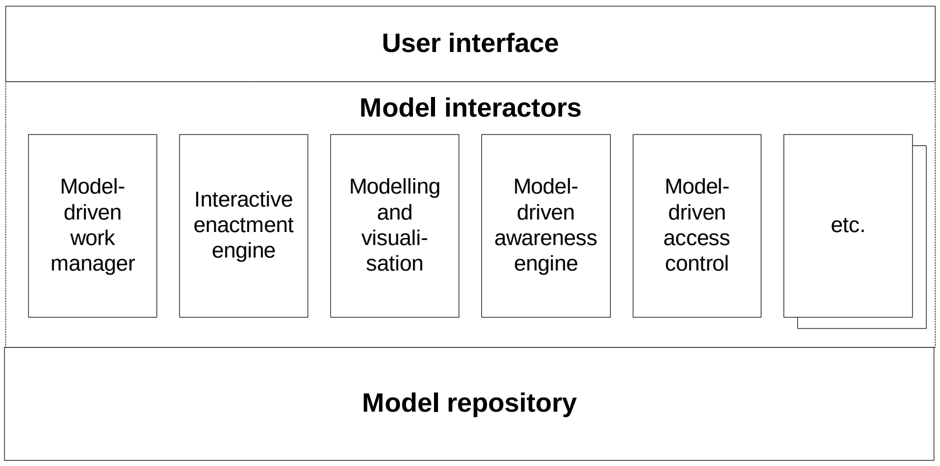

User interface
Model interactors
Model-driven work manager
Interactive enactment engine
Modelling and visuali-sation
Model-driven awareness engine
Model-driven access control
etc.
Model repository

Figure 27. Logical architecture of an interactive WMS.

Figure 27 shows a few model interactors, but the set is extensible. Any software that uses the models can be thought of as a model-interpreting component, whether interpretation is performed automatically, interactively or manually. Research should investigate the utility of workflow models in supporting different functions of the system, i.e. which interactors we can usefully include in the architecture. Examples include:

- Personal work organisation through worklists and workspaces for each workitem,
- Coordination through interactive enactment,
- Articulation integrated with rich visualisations in the model editor,
- Coordination through awareness, notifying users about the actions of others,
- Access control, so that each user's assignment to tasks influence which information is made accessible, on a need-to-know or a need-to-hide basis, subject to local policies,
- Communication support for collaborative tasks and negotiations, for informal handling of unforeseen situations etc.,
- Warnings when modelled rules are violated,
- Guidance for novice users, taking them step by step through unfamiliar procedures,
- Information management and workspaces for document-centric collaboration,
- Enterprise resource management integrated with workflow personnel allocation,
- Project management functionality, monitoring progress, time, money, and resources.

For each of these components the distribution of work between the system and its users needs to be studied. It is unlikely that one solution fits all processes, hence we need to develop customisable policies. Though adding details about access control, communication and resource management can make the model more complicated, these difficulties can be helped with visualisations that filter out dimensions from the model. Also, template policies can be offered for different process classes, and in many cases reused with little or no modifications. Anyway, the utilisation of workflow models to provide a wider range of context-sensitive functionality, removes the overhead of having multiple representations, and increases the users' benefits of keeping the models up to date.

The interactive architecture makes no separation between build time and run time. This follows the shift from input and output strings (Turing) to interactively generated streams [524]. Each component is seen as contributing to an integrated stream of model articulation and activation events, not as an input or output device. Though the model editor and the enactment engine are separate components, they have the same position in the system, that of using and updating the model. The modelling editor's visualisation capabilities can be a powerful activation tool, supporting manual coordination by giving users an overview of the current state of the process. In other words, the component conventionally thought of as the process definition tool [532], also plays a role in model activation. Conversely, users can articulate a process structure through a textual worklist, so conventional activation tools are suitable for process definition as well. Model interactors thus integrate articulation and activation. Figure 28 shows how the core concepts of interactive models (articulation and activation) extend those of workflow management, bridging the gap between modelling/definition and enactment[1].

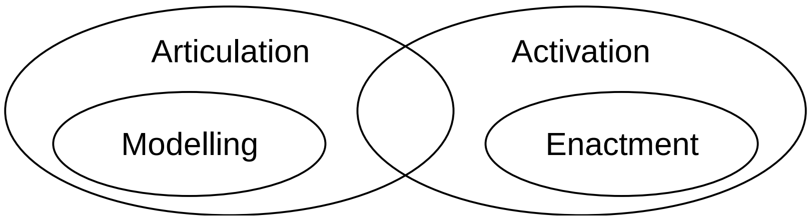


Figure 28. Core concepts in workflow management and interactive models.

4.4.1 Groupware Coordination Complements Workflow Enactment

When a workflow is loosely specified, the enactment engine provides little support. In these cases other interactors take over. Shared worklists [283] allow the users to communicate information about their work in the context of particular projects. The repository ensures that when one user e.g. declares a workitem finished, the updated status immediately becomes available to everyone else.

The *awareness engine* is another conventional groupware component [415, 428]. It provides information about the actions of others to each user within the context of her workitems. Since flows represent dependencies that must be coordinated, they are important *channels* for event notifications. The awareness engine thus utilises models to provide environmental feedback [207], propagating event notifications through the workflow structure. Only events relevant for the workitems that the user works on are forwarded. Presenting awareness information within the context of a workitem decreases the cognitive burden of interpreting this information. This also reduces the in-

[1] The figure shows the extension of phenomena referred to by the concepts [81].

formation overload problem often associated with asynchronous awareness mechanisms [141]. The detailed semantics of the awareness engine is presented in section 4.7.

Awareness, shared worklists and process visualisations may trigger opportunistic involvement [142], where users override the prescribed flow of work, e.g. to speed up the process or handle exceptions. For instance, a user may decide to start a workitem although not all of its required inputs are ready. When you are provided with information about the progress of work, you are better equipped to make such decisions. Although these actions violate the prescribed process rules, they may in some cases be tolerated by the system.

These groupware components support manual coordination through *mutual adjustment*, which Miers [349] pointed out as the main challenge for workflow research (cf. Figure 16 on page 61). Enactment and awareness engines both utilise the model to support coordination, the first by automated scheduling according to standard procedures, the latter by forwarding and presenting event notifications in the contexts where they are useful [416]. These interactors complement each other, handling different scenarios (sequential vs. concurrent work).

4.4.2 User Interfaces for Integrated Work Management and Performance

In WORKWARE, workitems are available from the work management tool. A *worklist* may present all items belonging to a project, or only the ones allocated to the current user. Meeting requirements for customisation, a query language lets users define their own worklists. Typically workitem status (new, ongoing, finished etc.), personal responsibilities (my tasks, other peoples tasks, tasks I participate in), project or work-package connection, and time considerations (overdue, deadline within the next week etc.) are utilised in these queries.

Each workitem is accessed through its *worktop*, which contains attributes that describe the item and links to relevant resources (tools, people, information etc.). A worktop is configured according to the particular workitem and the preferences of the user. Through this interface the users have access to a customisable selection of workitem services for planning and articulation (modelling, define new items), coordination (make decisions, change status), communication (email and real-time collaboration) and work performance (document management, desktop tools etc.). Services correspond to commands the user can invoke on the workitem. What each worktop includes is controlled by the interactive process model in a number of ways:

- Communication channels are included for all people participating on the workitem, as individuals as well as groups (e.g. mailing lists). When the users discover that more people need to be involved, they can add new roles to the process models and the communications channels will be automatically updated.
- Information modelled as resources to the workitem, are immediately available in the worktop. If a user adds a new document or an updated version, this is automatically reflected in the model, and available to other people.
- The description of a workitem shows all the properties that are currently defined. Each workitem can have local, user-defined properties.
- Navigation structures are generated based on the structures of the process models, e.g. to show sub-items or browse to the other end of a flow.

4.5 Detailed Activation Semantics

We will now look into the detailed definition of the modelling language and of the semantics that different interactors apply to the models. The *work management* tool provides basic means for users to articulate and share process models, but the automatic support that this interactor offers is minimal. The *interactive enactment engine* provides more automation, and the enactment rules fill a major portion of this section. But first, we formalise the syntax of the modelling language.

4.5.1 Workware Language Metamodel

Figure 29 shows the modelling concepts of WORKWARE in a UML class diagram. The attributes, operations and relations that influence upon the enactment semantics are included. Other attributes and relations are left out to improve readability. The interaction framework manifests itself through the introduction of users, roles and user interface components in the language meta-model. These concepts are needed to formalise the *interactive* operational semantics of the models. In addition to the modelling concepts *Workitem*, *Decision*, and *Flow*, we have included *Person* (users who fill roles as *Customer*, *Responsible* or *Participant* on a *Workitem*), as well as *Worklist* (a user interface component).

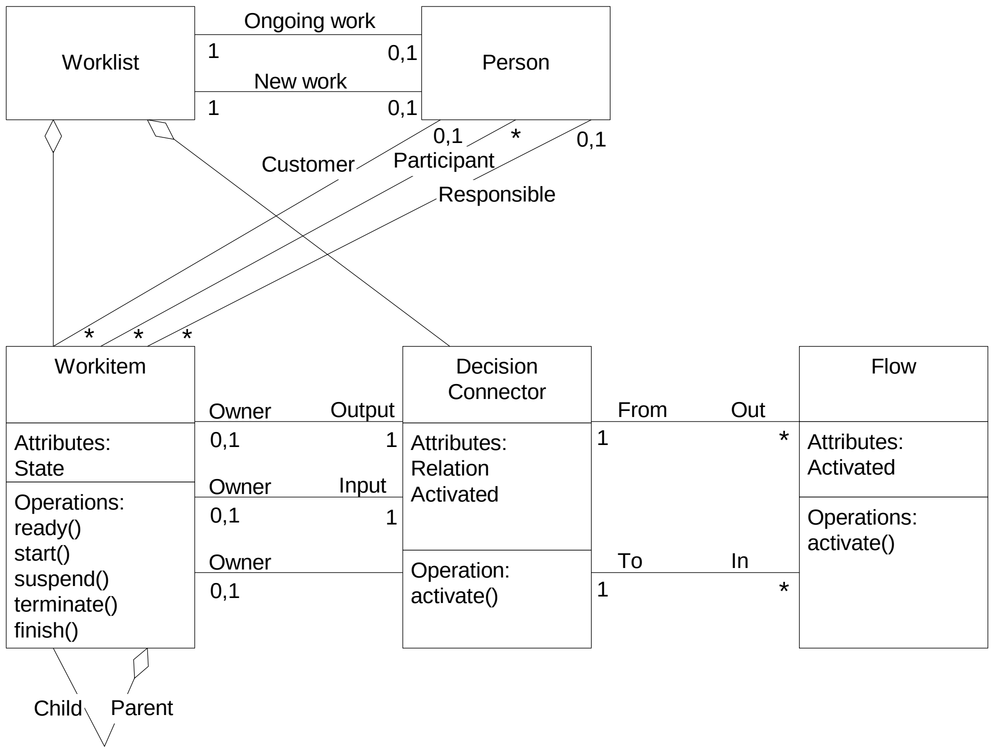


Figure 29. WORKWARE metamodel.

Each person has access to a number of worklists with different selection criteria. Here we have included the standard lists *New work* and *Ongoing work*. The operations in the metamodel are triggered by other objects or by users invoking services. Figure 29 shows that each workitem has one input and one output decision connector, as described above. Decisions have access to their surrounding workitem through the *Owner* role. Each flow relates exactly one source (*from*) and one destination (*to*). Both are *decisions*. Flows to and from workitems are connected to their input and output decisions.

The relationships *Responsible*, *Customer* and *Participant* refer to types of roles that *persons* can play in a workitem (section 4.2.6). In the graphical modelling editor these role assignments are done through *actor resources* (Figure 21 on page 84), but in WORKWARE the relationships are traversed and short-circuited so that only the resulting relations (which person fills which role) are represented. This simplification better matches the immediate nature of work performance, while indirect relationships in the model editor provide better support for planning, reuse and resource management. Each workitem has one responsible, at most one customer, and any number of participants

4.5.2 Activation by the Work Management Tool

The work management tool provides worklist and worktop user interfaces, giving access to services provided by the system or integrated from other tools. This section describes the semantics of the workitem lifecycle as it is controlled through these interfaces. The component can function independently of the enactment engine. Since it offers no automatic support, users must trigger all state transitions through coordination services. In the metamodel, these services manifest themselves as operations of the workitem class. Figure 30 shows this interpretation, involving the following states:

- *New*, not yet started. These items are put in the *New work* worklist of the responsible user (as defined by the invariant rule of the state).
- *Ongoing*, work is being done on this item. These items are put in the *Ongoing work* worklist of the responsible user.
- *Suspended*, signalling e.g. that the work is pending some input in order to be resumed. At any time, the participants may resume the work.
- *Finished*, signifying normal completion.
- *Terminated*, representing abnormal termination or deletion of a workitem. Any workitem not already *finished* can be terminated.

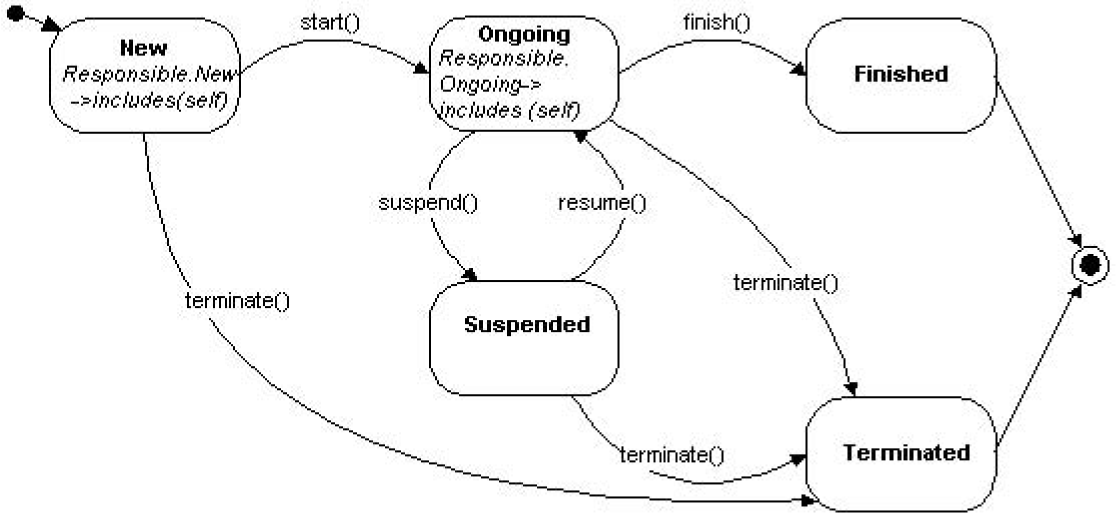


Figure 30: Work items interpreted by the work management interactor.

The worktop of an item gives the users access to a different set of coordination services in each state. Coordination services invoke state-changing methods on the workitem, so the outgoing transitions from each state define which coordination services are available. For instance, when a workitem is ongoing, users may suspend, finish or terminate it. Figure 31 shows the scope of interactive model activation, from fully automated to manual schemes. Interactive schemes occupy the middle ground between these two extremes. As we have already discussed, user involvement can be reactive (triggered by the software) and proactive (triggered by the users). Past research [283] shows that a work management tool that enables personal work organisation and sharing of work-related information can be useful by itself even without enactment support. This scheme demonstrates the utility of *manual model activation*. But it has the disadvantage of requiring users to do all the scheduling and status reporting manually, hence we included more automated coordination support through interactive enactment.

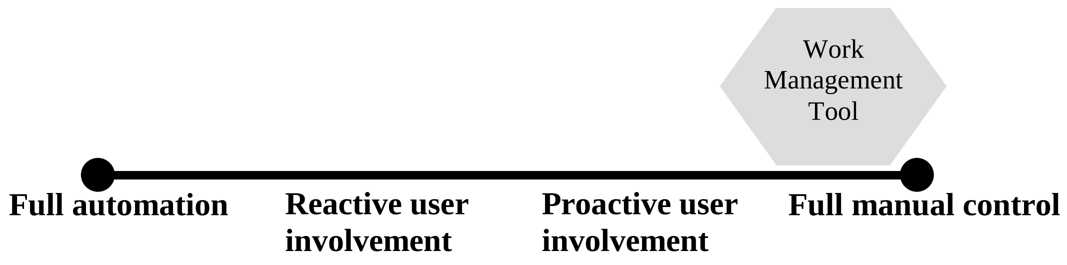


Figure 31. The work management tool in the interactive activation spectrum.

4.5.3 Activation by Interactive Workflow Enactment

In WORKWARE the flow of work is propagated by updating the status of workitems, and by activating flows and decision connectors. While the work management tool left all of these tasks up to the users, the engine also automates a number of enactment rules. The model below shows the enactment state transitions for workitems[2]. The states and transitions defined by the work management tool are shown in normal font, while the rules introduced by the enactment engine are printed in bold. Comparing this diagram to Figure 30, we see that the state *New* has been split in two:

- *Waiting* means not ready to start due to pending input flows. Users can override this rule by explicitly declaring the item ready or started. Items in this state are not shown in the responsible user's *New work* list.
- *Ready* to start. Users must still explicitly start the work manually. Items in this state are shown in the responsible user's *New work* list.

The rationale for requiring manual start of workitems is given by [142]. This scheme allows us to separate the tasks that really are ongoing from the ones that the system says are ready to start. This information is important for a number of reasons. A ready task may be more easily re-allocated to another person than one where the work has already started. Items that remain ready for a long time without being started, may point to a

[2] Conditions are placed in [ ] parentheses. Triggered actions follow after the /. This syntax follows the standard UML Object Constraint Language OCL [517]. A complete UML model would also include mappings between messages, operations, methods etc., here hidden for readability. When operation names are used as events triggering transitions, they refer to the effect of the operation. When an operation name is used in an action triggered by a transition, it refers to the invocation of the operation.

large backlog of the person responsible, and thus inform later allocation decisions. UML Action Semantics [382] model distributed, concurrent and abstract execution behaviour. Noting that traditional programming languages over-specify sequence, complicating a flexible assignment of actions to components in a distributed execution environment, the UML action lifecycle also includes both *Waiting* and *Ready* states. In addition to these new states, enactment rules connect the behaviour of the workitem with its inputs and outputs:

- When the *input* decision is activated, the item goes from *Waiting* to *Ready*.
- When a suspended item receives an *in flow* through its *input*, it is restarted.
- When the output decision of an item receives all the flows it is waiting for, the item finishes automatically.

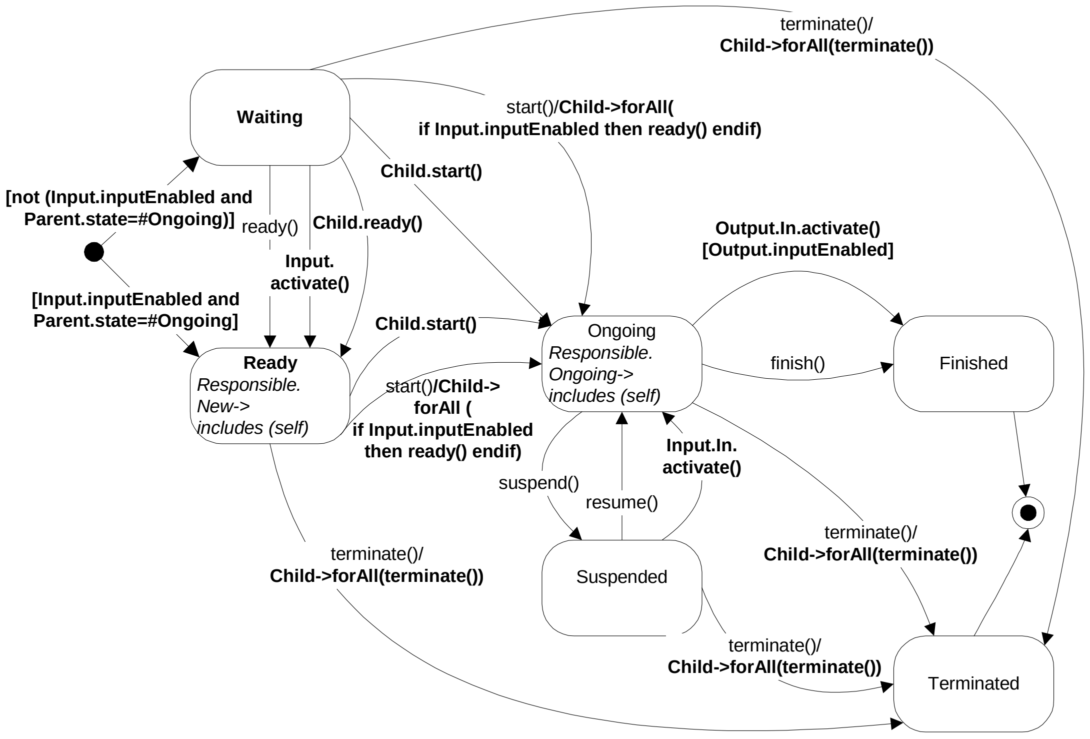


Figure 32: Workitems interpreted by the enactment engine.

Workitem Decomposition

There are also a number of rules that follow the work breakdown structure. They define how children affect their parents and vice versa. These operational rules are generally implemented in order to ensure that children are performed as parts of their parents. This is a basic assumption in the engine's interpretation of the models, but users may invoke coordination services that violate this constraint, causing an exception.

- When a child of an item becomes *Ready*, the parent also becomes *Ready*.
- When a child is started, the parent is automatically started as well.
- When a parent is started, all children not waiting for input flows are made *Ready*.
- When a child finishes, and activates a flow to the output of the parent, if the output has received all necessary flows, the parent automatically finishes as well.
- When a parent is terminated (e.g. withdrawn), all its children are terminated as well.

These rules enable users to articulate different scenarios for the work breakdown structure: If they want the parent to finish automatically when all children are finished, they must add flows from the sub-items to the output decision of the parent item (as in Project A in Figure 33 below). If there is some work to do that is not articulated as sub-items, or if the person responsible wants to control the scheduling of work herself, the output should be left without input flows (as in Project B in Figure 33). Note also that while in Project A the enactment engine will trigger the sub items in a waterfall sequence, Project B gives the engine very little to do, as the sequencing of work is left to the users utilising the work management interfaces.

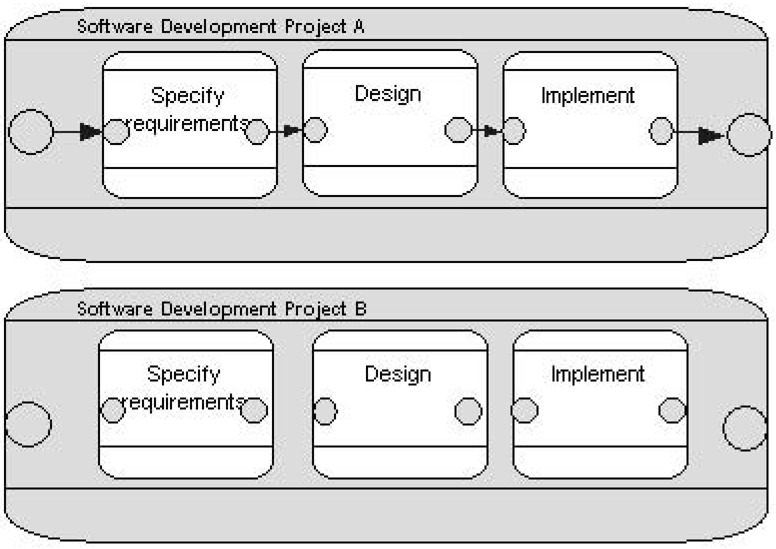


Figure 33. Two process models with different degrees of structure.

Workitem Lifecycle

The initial state transition diagram only referred to the enactment phase of the lifecycle of a workitem. The planning, articulation, or definition, of a work item model, is not represented in Figure 32. For emergent workflows, definition and enactment are intertwined. For plan changes, we use this simple state transition rule:

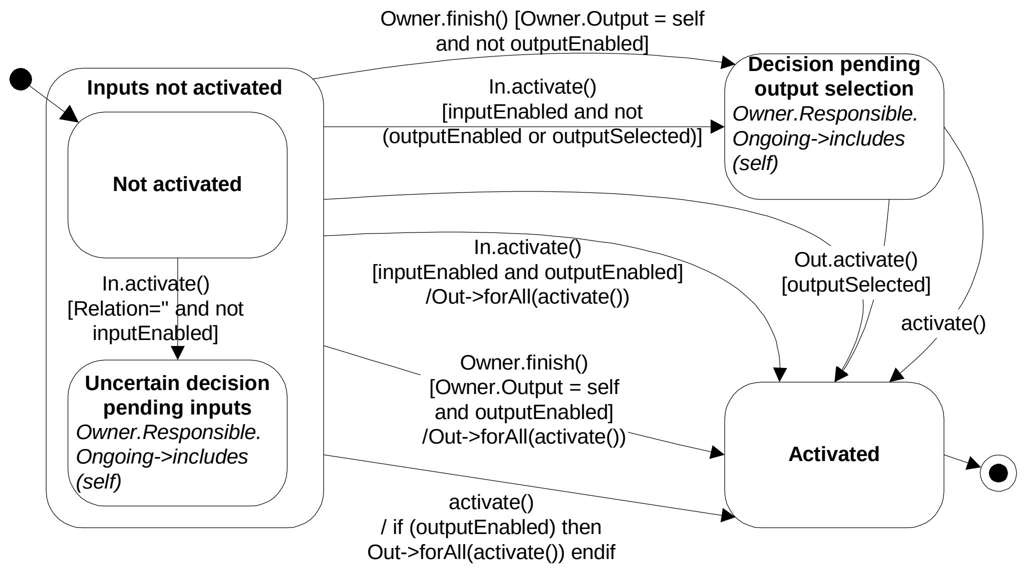
- When a new workitem is added, its default state is *Waiting*, unless its *parent* is *ongoing* and the *input* of the new item is *enabled* (defined below), in which case the new item is set to *ready*.

Rules and mechanisms for extending the enactment scheme are discussed in section 4.6.

Decisions

A decision connector represents an issue to resolve regarding the flow of work. Some decisions are automated while others are resolved by users. The position (*owner*) of a decision connector defines which user is responsible for making the decision. A detailed model of the dynamic behaviour of a *Decision* is presented in Figure 34. The two main states are *activated* and *not activated*. They are reflected by the Boolean attribute *activated*. In addition, there are two states that arise from the interactive nature of the enactment scheme. Both represent situations where the decision cannot be automated, and human involvement is needed. Through the *fallback mechanism,* these decisions are added to the worklist of the user responsible for the owner work item. This mechanism thus implements the user involvement scheme presented in Figure 26:

- For *join* decisions, the problem is when to proceed (activate the decision).
- For *fork* decisions, the problem is which alternative(s) to choose.


Owner.finish() [Owner.Output = self and not outputEnabled]
Inputs not activated
Not activated
In.activate() [inputEnabled and not (outputEnabled or outputSelected)]
Decision pending output selection
Owner.Responsible. Ongoing->includes (self)
In.activate() [Relation='' and not inputEnabled]
In.activate() [inputEnabled and outputEnabled] /Out->forAll(activate())
Out.activate() [outputSelected]
activate()
Uncertain decision pending inputs
Owner.Responsible. Ongoing->includes (self)
Owner.finish() [Owner.Output = self and outputEnabled] /Out->forAll(activate())
Activated
activate() / if (outputEnabled) then Out->forAll(activate()) endif

<u>Invariant rules for Decision (derived attributes):</u>

fork =	Out->size >1
join =	In->size >1
inputEnabled =	(In->forAll(Activated)) OR ((Relation = 'XOR') AND (In->exists(Activated)))
outputEnabled =	(Relation = 'AND') OR (NOT fork)
outputSelected =	(Out->forAll(Activated)) OR ((Relation = 'XOR') AND (Out->exists(Activated)))

Figure 34. Decisions interpreted by the enactment engine.

A special decision making form facilitates user interaction with individual decisions. Some decisions, e.g. AND-forks and XOR-joins, can always be automated. In order to make the model readable, we have defined a number of derived attributes:

- *fork*, whether the connector has multiple output flows,
- *join*, whether the connector has multiple input flows,
- *inputEnabled*, whether the required input flows have been activated,
- *outputEnabled*, whether the engine is able to automatically decide which outputs should be selected,
- *outputSelected*, whether the maximum number of allowed outputs have been selected (one for alternative selections, all for concurrent branching).

All these properties are derived from the structure and states of surrounding model elements. This enables users to easily add and remove both input and output flows without concerning themselves whether the connector was originally defined as a *join* or a *fork*. Not only does this enable a simplification of the language (no need for separate constructs for joins and forks); it also increases the fluidity of process articulation. In addition to the flow activation actions triggered in Figure 34, Figure 32 defines the effects of a decision connector upon its owner workitem.

Flow

The state transition diagram for flows is depicted in Figure 35. Similar to decisions, flows go from not activated to activated. In the simple case, a flow can be activated explicitly be the user. In addition to human intervention, the enactment engine automates some state transition rules (mostly defined in the diagram for decisions, Figure 34):

- When a decision with one output flow is activated, the output flow is also activated.
- When an AND decision is activated, all output flows are also activated.
- When the target decision (*to*) is activated before the flow, a *flow violation* may occur. Here we have an exception where the model does not match what is actually happening. For now we just capture the exception, later we will look at how the flow dependency can be coordinated by the awareness engine in this situation.

A flow violation may be the final state of a flow if the exception is never resolved (e.g. by the activation of the flow or its origin). A flow may also end up never being activated at all, e.g. because another flow triggers its destination connectors, making this flow superfluous. Another case is when the source decision is made, but another alternative is selected (another flow is activated).

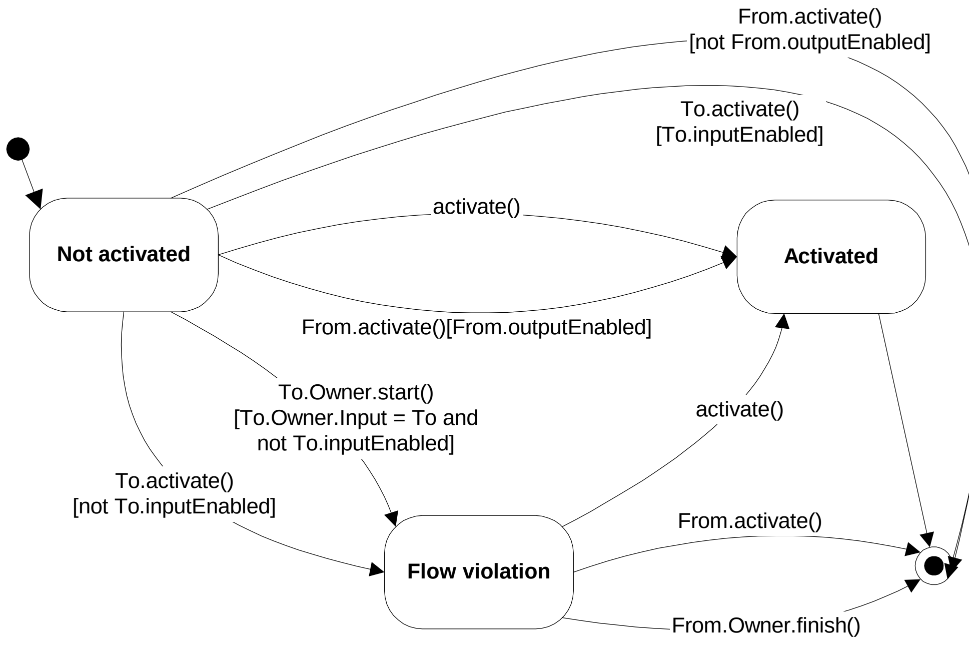


Figure 35. Flows interpreted by the enactment engine.

Figure 36 shows the scope of the interactive workflow enactment engine in the activation spectrum presented in Figure 31. It illustrates that different distributions of decision making between the engine and the users are supported, depending on the current state of the workflow model and the actions of the users.

4.5.4 Combined Enactment Semantics

Now that we have explored the basic enactment rules associated with workitems, flows and decision connectors, we can look at how these rules interoperate to support typical workflow scenarios. Such scenarios can be represented as UML interaction sequence diagrams, involving users playing roles on the workitems as well as the modelled objects. The scenarios illustrate that WORKWARE's activation mechanisms are capable of handling models with varying degrees of structure.

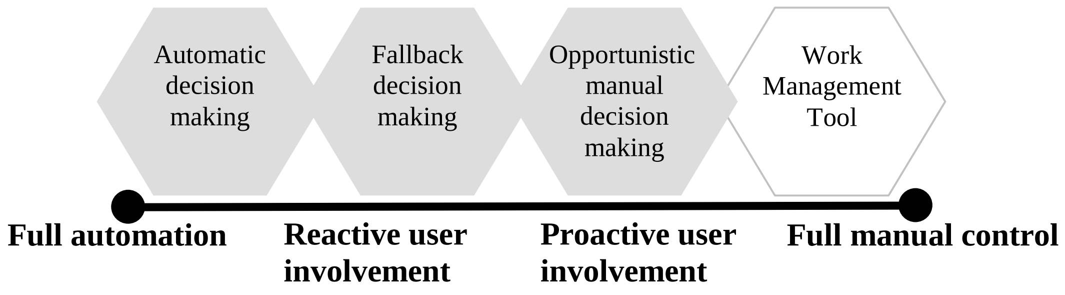

Automatic decision making
Fallback decision making
Opportunistic manual decision making
Work Management Tool
Full automation
Reactive user involvement
Proactive user involvement
Full manual control

Figure 36. The workflow enactment tool in the interactive activation spectrum.

Workitem Creation, Start and Suspend

Figure 37 shows a fragment of the interaction among a user, a workitem with input decision connector and two input flows. The user creates the workitem, which is then made ready by the activation of an input flow (*In1*). Later the user suspends the item, and finally the activation of another input flow causes it to restart. This diagram is simplified in a number of ways. It does not show the user interface components through which the user interacts with the modelled objects, and it only reflects some aspects of a short segment of the lifecycle of a typical workitem. The sequence also involves just one user. The enactment of a complete W0ORKWARE model, involving multiple users, user interfaces (worklists and worktops), and modelled objects (workitems, flows, decisions etc.), would be too large an arrangement to present this way.

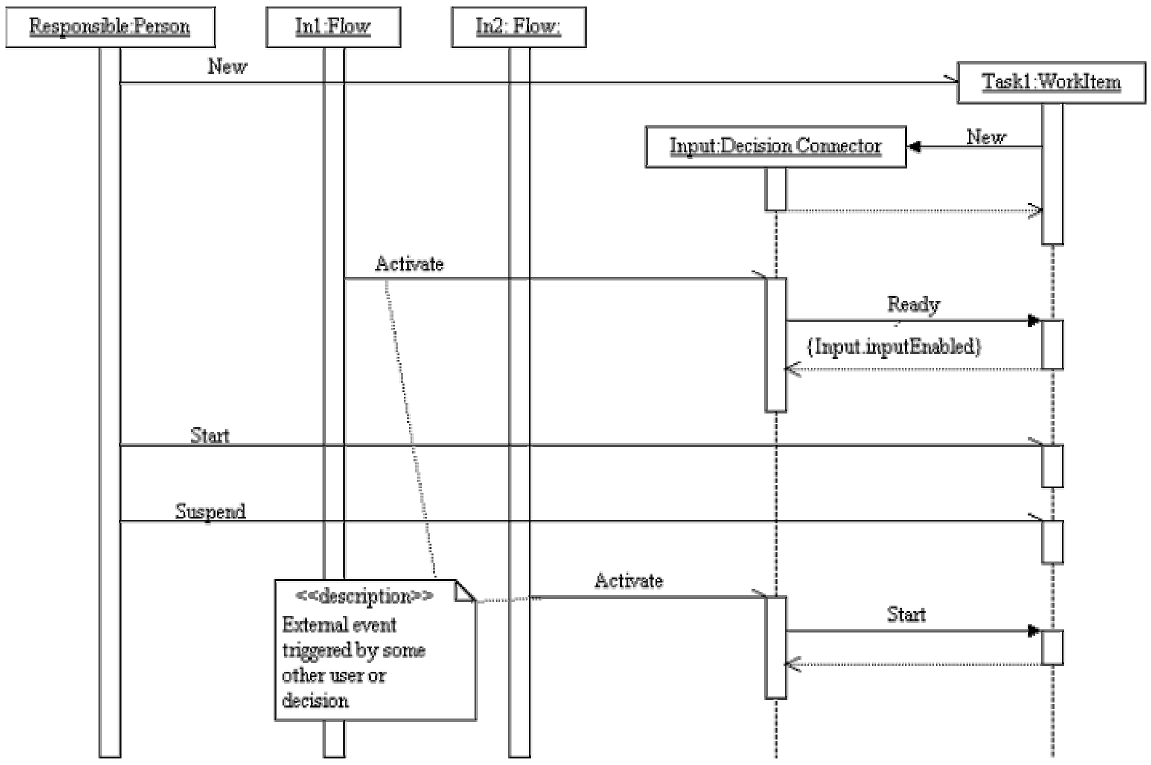


Figure 37. Interaction between user and model in starting and suspending a workitem.

Workitem Completion

Figure 38 shows the interaction involved in starting, carrying out the work, and finishing a typical workitem with one input and one output flow. Two alternative completion scenarios are included, one where the user decides to activate the output flow before the workitem is finished, and another where the output flow is automatically invoked when the task is declared finished by the user. The details regarding what happens during the actual performance of the work, is only represented by a single example message in the in the model.

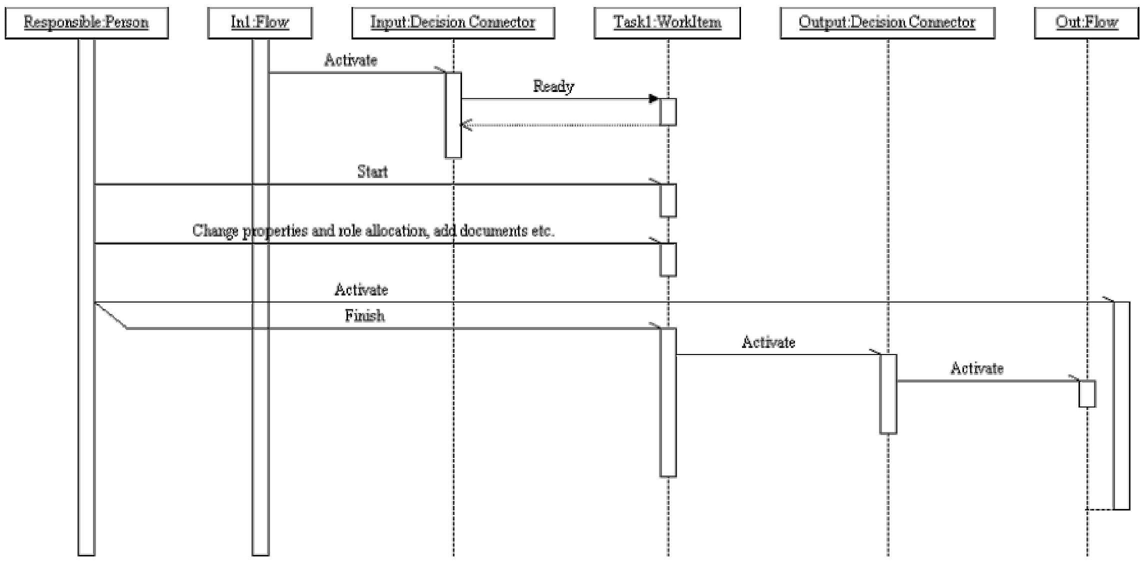


Figure 38. Interaction in two typical workitem activation cycles.

Unstructured Workitem Collections

The process enacted in this scenario is "Software development project B" from Figure 33. In this model the parent workitem has three sub-items, but there are no flows that determine their sequence. This means that the users must control the sequence of the sub-items. In the depicted scenario, requirements are completed before design and implementation starts, but the two latter are performed partially in parallel. In order to make the diagram readable all decision connectors are hidden.

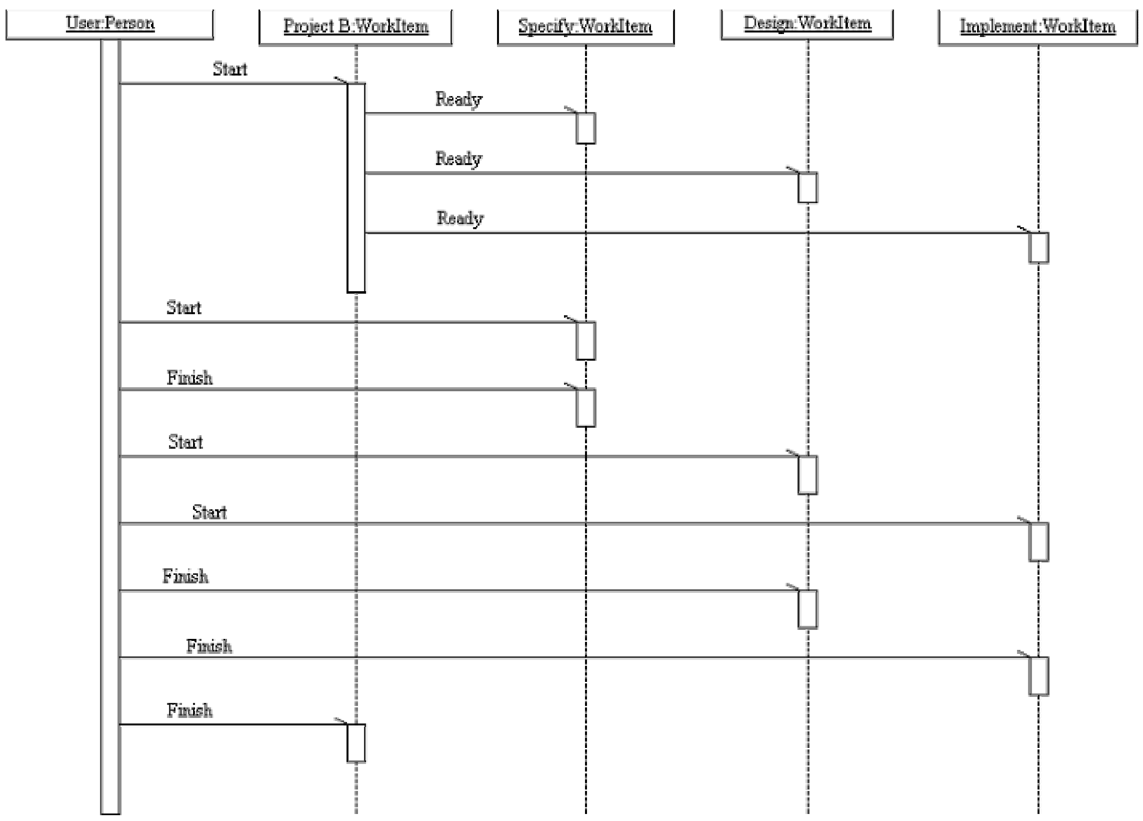


Figure 39. Enactment of unstructured workitem collection.

Structured Workflows

The process enacted in this scenario is "Software development project A" from Figure 33. The diagram in Figure 40 looks very much like the one in Figure 39, but the sequence of the signals are a bit different, and the automated support increased:

- The sequence specify→design→implement is triggered by the enactment engine,
- The project is automatically finished when the last sub-item is finished.

The flows and connectors that make these automatic actions happen, have been left out of the diagram to make it readable, but they follow the state diagrams shown earlier in this chapter and the more detailed interaction scenarios of Figure 37 and Figure 38.

Dynamic Replanning

Figure 41 shows a variation of the structured software development scenario (Figure 40). During requirement specification, the participants decide to add a separate *Test* workitem after the implementation. After the implementation, however, the users discover that sufficient testing has already been done as part of the implementation work, and thus the *Test* item is terminated. Since termination is an abnormal action, it does not result in activating the out flow. Consequently, the project does not auto-finish as it did in Figure 40, rather in this situation we require the responsible user to manually verify and declare the overall project as finished.

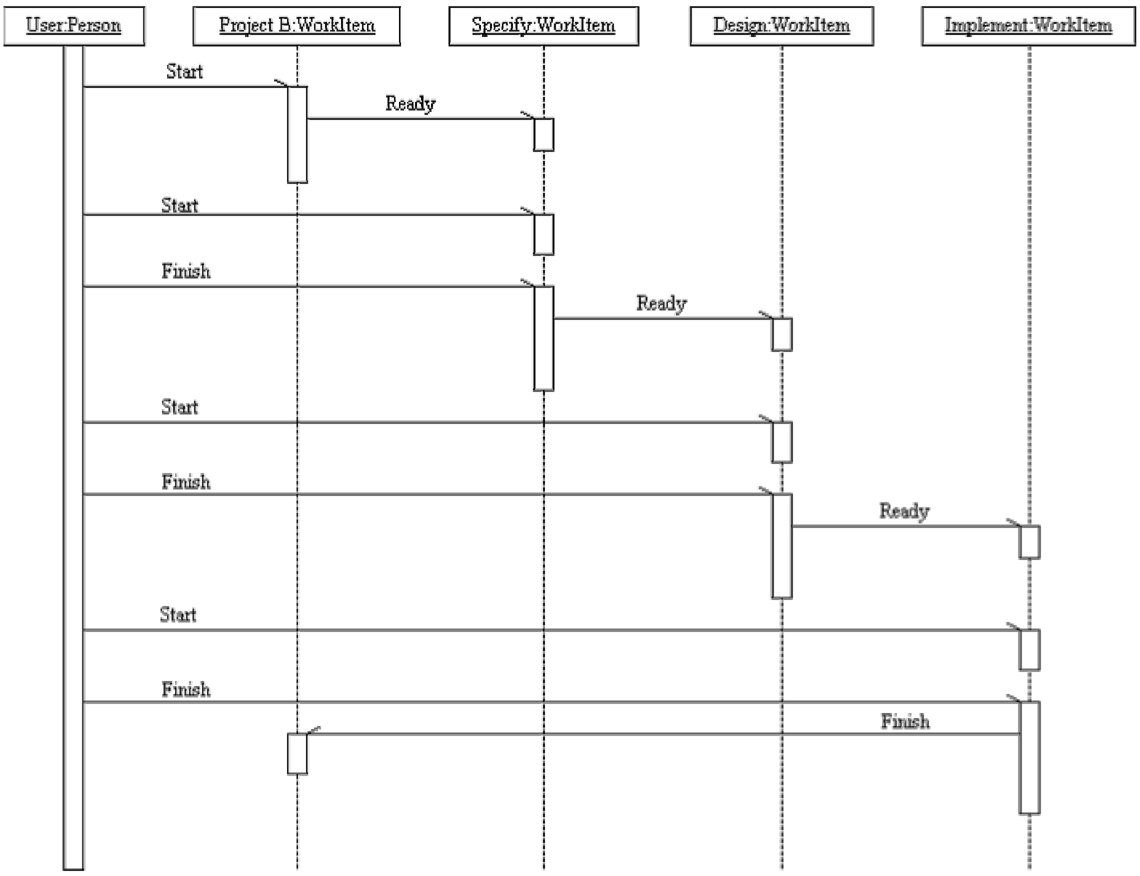


Figure 40. Enactment of structured workflow.

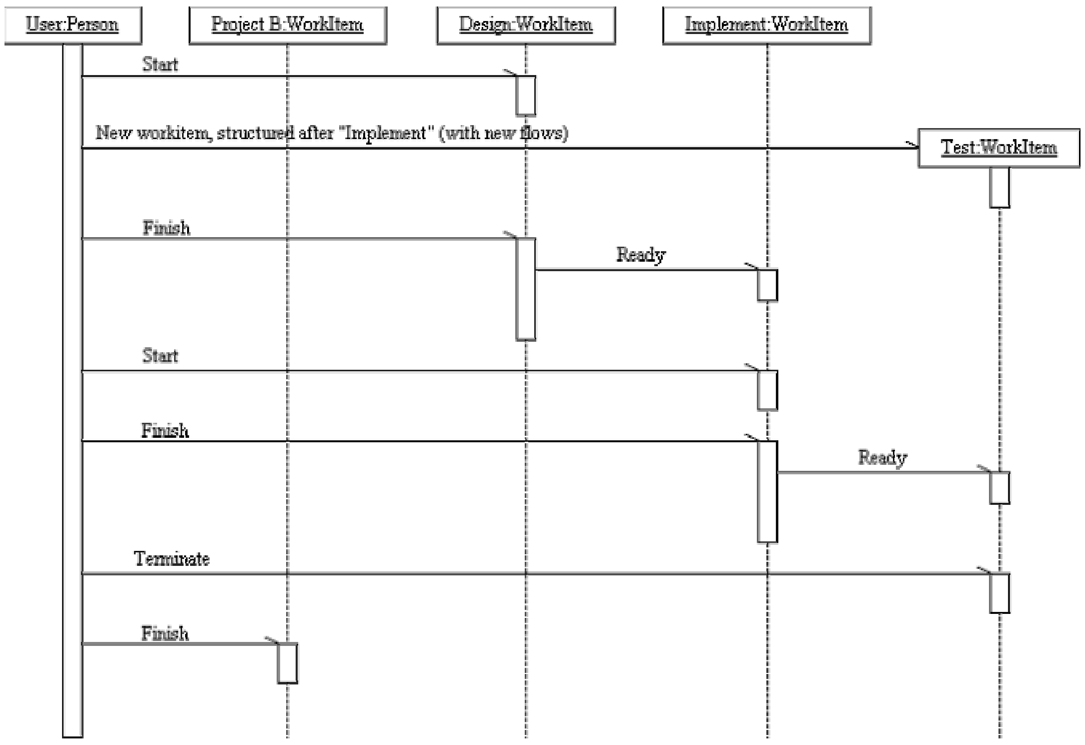


Figure 41. Dynamic replanning of a structured workflow.

4.5.5 Metamodelling Interactive Languages with UML

Most conceptual modelling languages are specified in a formal, symbolic manner, e.g. in a logic notation or Backus-Naur form [482]. Over the past couple of years, flexibility requirements have led to the development of systems that allow their languages/metamodels to change [138, 492]. Such reflective systems often apply a graphical notation in their metamodelling, in order to increase user-friendliness. This trend has spread also to UML [381] and to workflow management [551], where languages are specified in UML class diagram metamodels. These approaches to language definition primarily deal with the *syntax* used for articulation. With interactive models, we are also interested in the *activation semantics*. Therefore we have added state transition and interaction diagrams to our metamodels. Workflow standardisation efforts [532] have also used state diagrams, but not with as much detail. Another approach is to use Petri Nets for integrating metamodelling into process modelling [554]. However, some innovative features in WORKWARE have lead to a different use of UML for metamodelling:

- *Interactive enactment*, hence users and user interface concepts are included in the language models and interaction sequences. Partially structured models and addition of new objects illustrate the open and evolving nature of interactive models.
- *Multiple model interactors*, hence there is a need for multiple, incomplete diagrams reflecting the semantics imposed by each of the components, and mechanisms for integrating these views. UML tools are currently weak in this area. The presentation used here was an incremental one, e.g. the enactment engine semantics (Figure 32) for completeness were presented alongside the work management semantics from Figure 30. This effort revealed the close interdependencies among the interactors. For instance, we saw that the enactment engine increased the usability of the work management tool by separating *new* workitems into *ready* and *waiting*.
- *Holistic enactment semantics* implies that the interpretation of a model element depends on the other elements in the model. Here the navigation constructs from OCL [517] proved valuable. By browsing relations, complex interdependencies can easily be defined, e.g. *Owner.Responsible.Ongoing work->includes(self)* to declare that this item should be part of the worklist of the person responsible for it. Interaction sequence diagrams also illustrate the many interdependencies among model elements in typical activation scenarios. These diagrams quickly become too large, even for very simple examples. Consequently, some aspects of the models had to be left out. To enable this kind of metamodelling, constructive composition and complexity reduction (simplified view definition) of interaction sequence diagrams should be better supported in UML modelling tools.

The open and holistic character of the semantics also makes it difficult to define any model completely. For instance, in Figure 32, the interdependencies along the work breakdown structure just refer to children, not to parents. The influence of the parent is modelled in that parent's state transition diagram (the same diagram for another object instance). So in a "complete" model of the workitem state transitions, you would model every one of these transitions twice, once as actions triggered by a transition and once as transitions in their own right. Such duplication makes the model cumbersome to maintain, so we decided to model everything once, with all references to children and none to parents. In all, the use of UML class and state transition diagrams with OCL rules has worked reasonably well for metamodelling in this basic specification. As we move on to instance-specific and property-controlled behaviour, the class-oriented foundation of UML becomes more problematic.

4.6 Extended Activation Semantics

The enactment semantics of the WORKWARE language can be enriched in a number of ways. Although this thesis focuses on the process aspects of the models, other aspects, e.g. resources, products, information and organisational structures, can also be activated. In the process domain, we can increase the automated support by enlarging the vocabulary of decisions.

4.6.1 Decomposed Decisions

When the ports in APM were generalised into the decision concept for WORKWARE we lost the ability to model multiple entry points to a workitem. While each action in APM could have multiple input and output ports, we simplified this to only one for workitems. The special case of multiple entry points could be easily handled by decision decomposition, as shown in Figure 42.

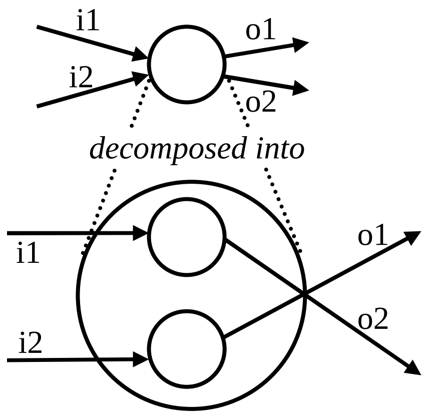


Figure 42. Decomposed decision.

Decomposition also caters for more complex triggering rules, for instance regarding the logical relations represented by the decision connectors. An example is shown in Figure 43, representing the rule: *When i2 is activated, if i1 has been activated, trigger o1, else trigger o2.*

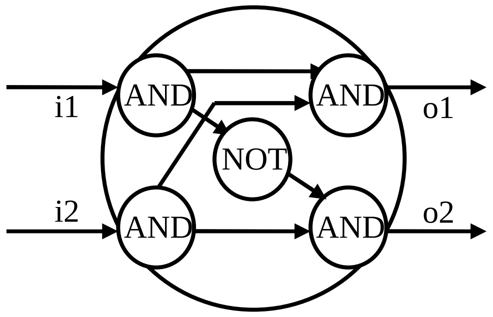


Figure 43. More complex rules for automated decision making.

The *NOT* decision connector is introduced here to facilitate such rules. The output flow(s) of a NOT decision is active until the input is activated, at which time it becomes inactive. This connector also handles the *escalation* pattern [531], invoking a process in case a constraint is not met. We do not expect all end users to articulate such detailed enactment rules. In many cases it is simpler just to make the decision manually. The aim of these mechanisms is to provide powerful constructs that allow process experts to create reusable process templates with higher degrees of automation.

4.6.2 Parameterised Decisions

Data driven enactment rules allow different paths to be chosen depending on the input data. This requires that flows can carry data values. WORKWARE's metametamodel (presented in Chapter 5) allows flows and other objects to carry user-defined properties. An example use of such a scheme is presented below.

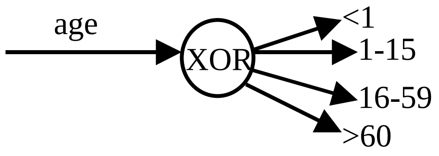


Figure 44. Parameterised decision connector.

4.6.3 Timers

Timing and scheduling of workitems are important aspects of project management. The default properties of each workitem include a due date and expected start date. These properties are used by the work management tool to identify delayed tasks, and e.g. present them in separate worklists or with special symbols. But WORKWARE also offers constructs that allow users to model time-dependent processes. *Timers* are decisions that trigger their output flow(s) at a given point in time. Timer was included as a first class primitive in APM. Figure 45 shows that timers can be absolute or relative.

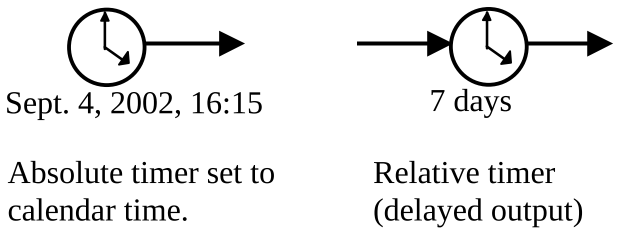

Sept. 4, 2002, 16:15
7 days
Absolute timer set to calendar time.
Relative timer (delayed output)

Figure 45. Absolute and relative timers.

Timers can serve a multitude of purposes, including:

- *Scheduled tasks*, whose input decisions are timers. Typical examples are meetings and reporting workitems.
- *Milestones*, signifying that a set of workitems contributing to a joint deliverable, should be finished before a given date. Milestones are typically major assessment points for progress and resource consumption in a project. A template for milestone decisions is shown in Figure 46. It gathers all the inputs, and produces one output

when the inputs are finished, and potentially another output that trigger exception handling if the milestone is delayed.

- *Exception handling* and corrective actions can be triggered by the absence of some flows (NOT connector) at a given time. The degree of detail put into modelling such exception handling, is left to the user organisations to decide. As we have seen, WORKWARE offers a number of standard mechanisms for keeping users informed about the progress of workitems, so detailed exception handling can also be left to the users' discretion.

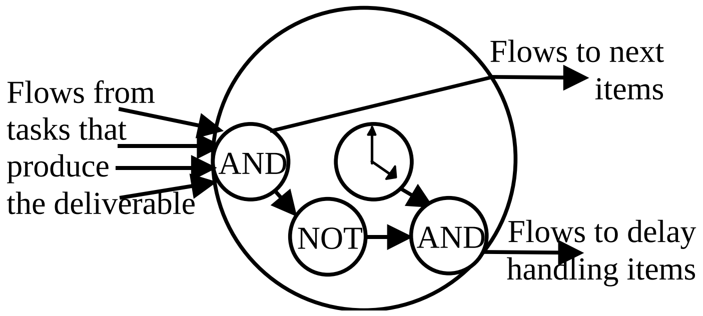

Flows from tasks that produce the deliverable
AND
NOT
AND
Flows to next items
Flows to delay handling items

Figure 46. Milestone as a timer with exception handling.

The case studies surveyed in Chapter 2 show many examples of explicit representation of work processes, but the processes of managing the work is seldom articulated. It has been noted that workflow systems can serve as "instruments of domination" [92], enabling some actors to control the work of others. This could be part of the explanation why some managers are reluctant to articulate their own work processes, but eager to articulate the processes of their subordinates. However, in knowledge intensive industries, management by direct supervision and standardisation of processes is often replaced by coordination through mutual adjustment, hence management/coordination is an integral part of the performance of work. In the case studies of the EXTERNAL project (presented in Chapter 8), we discovered that management and exception handling processes were easier articulate as generic templates than the core, knowledge intensive tasks.

4.6.4 Workitem Automation

In addition to coordination and work management support, most workflow systems provide some means of full or partial automation of atomic workitems For instance, web services can be integrated in a process in this manner [99]. In APM *invoked software tool* and *software agent* resources signify that the action is performed entirely by an external software component. WORKWARE utilises the presence of such resources to simplify the enactment semantics of the work item:

- *Start()* triggers the invocation of the tool and sets the status of the work to ongoing.
- Fully automated items, that require no human interaction during their performance, are automatically started when they become *ready*, and finish automatically upon completion of the invoked tool service. This is articulated by adding the property *AutoStart* to the workitem in question and by adding a flow from the input connector to the output connector so that it automatically finishes. In the state diagram of the

enactment semantics for workitems (Figure 32 on page 98), this cause the addition of a transition from *ready* to *ongoing* conditioned by *[AutoStart]*.

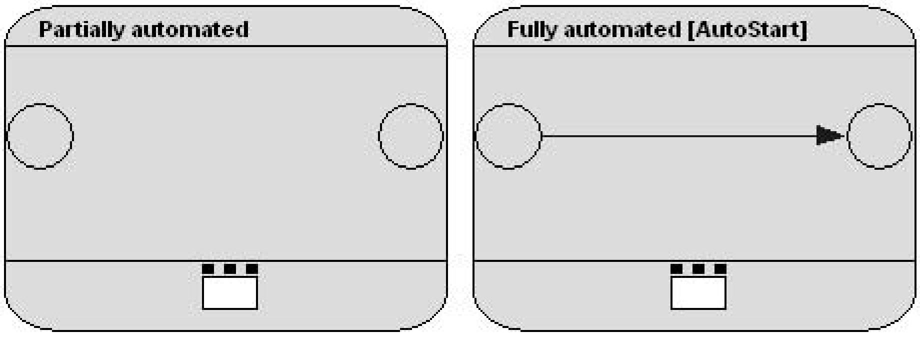


Figure 47. Partially and fully automated work items.

4.6.5 Reflection and User-Defined Activation Rules

Although the model activation semantics presented in this chapter handles a multitude of different scenarios ranging from loosely defined evolving processes to completely structured and automated ones, the case studies from Chapter 2 warns against assuming that a mechanism for process enactment can ever be complete. Consequently, the requirements state that the modelling language and its semantics should be able to evolve and support different interpretations. This subsection looks at various ways of enabling users (primarily process experts) to extend, alter and complement the activation rules.

Extensible State Sets

A first step towards user-defined activation semantics is to allow the state sets of the basic constructs to be altered. This is supported by WORKWARE's metametamodel, which will be presented in the next chapter. Allowing users to define new states is sufficient for manual activation, but not for automated support. Rules for how these states can be reached and left must also be defined. The enactment state diagrams presented above document the implementation of WORKWARE components, but they are not actively used during execution. In the future, UML virtual machines [411] might remedy this situation, but for now we are left with simpler means.

Event Condition Action (ECA) Rules

The transitions among states in the activation state diagrams are directly represented as Event-Condition-Action rules. By adding the guard condition

State = #FromState

and the action

State = #ToState

to these rules we capture the whole semantics of the transition. Previous research has shown that ECA rules yields inadequate performance for high-volume production workflow [98]. Consequently these rules have been utilised to control exception handling rather than normal enactment in adaptive WMS [98, 319, 320]. For emergent, knowledge intensive processes this scalability problem is not as severe, and ECA rules seem feasible for normal enactment as well. Indeed, emergent workflow is about treating exceptions and evolution as the rule of the game, rather than infrequent accidents.

Enactment Policies

In order to utilise ECA rules in WORKWARE, we need a rule-editing interface for process experts in the organisation. End users may utilise simple customisation functionality, e.g. selecting among a set of possible behaviours (policies), for entire processes as well as for single elements. Enactment policies provide users with a simple way of configuring the semantics of model elements. *Autostart* as described above is one such policy. Policies are attached to workitems and other objects as properties. Table 4 shows some sample enactment policies and their representation in ECA rules (in OCL syntax).

Enactment policy	**ECA Rule**
Autostart	[state=#Ready]/start()
Autostart children	[state=#Ongoing]/Child.forAll(if input.inputEnabled then start() endif)
Repetitive (restart)	[state=#Finished] : services->includes(start)[3]
Repetitive as new instance	[state=#Finished] : services->includes(clone)
Loop as cloning of workitem (rule for input connector)[4]	activate()[state=#Activated and Owner.state=#Finished] /Owner.clone()
Loop as restart of existing workitem	activate()[state=#Activated] /Owner.setState(#Ongoing)
Once only (automated items)	[state=#Ongoing]/finish()[5]

Table 4. Enactment policies.

Deontic Rules

Rule modelling in APM [90, 91] is supported by the deontic rule modelling language DRL [284]. DRL rules have this syntax:

> *when* trigger *if* condition *it is* deontic *for* role/actor
> consequence *else* another consequence

The *trigger*, *condition* and *consequence* parts of these rules correspond to event, condition and action in ECA rules. The *deontic* operator determines the strength of the rule. It is either *obligatory*, *forbidden*, *recommended*, *discouraged*, or *permitted*. Deontic operators thus allow human judgement in model interpretation. Deontic operators have a role in rules, but also in other model elements. For instance, it makes sense to denote a *flow* dependency as *obligatory* or *recommended*. The *flow violation* state in Figure 35 should not be allowed for obligatory flows. In model interpretation, deontic operators are most useful in general templates and organisational procedures, but they may also articulate the local commitments of process participants. In order to support models with varying degree of structure, the framework also needs to reuse deontic operators. For instance, a default deontic operator can be specified for a process, and applied to all of its components. The designation of role/actor for which a rule applies, can also be derived from the process model. A rule that is applied to a workitem by default applies to the responsible person for that item. Operationally, deontic rules should be activated

---

[3] *Services* here refer to the set of services available in the user interface for this object.

[4] Here the arrival of a flow to the input of an already finished workitem, generates a new copy (iteration) of the item.

[5] An alternative way of articulating "Once only" was shown in Figure 47.

by a *guiding and warning engine* in cooperation with the enactment service. This engine should guide users when they are working on a task, and issue warnings to various actors when rules are violated. Forbidden and obligatory rules may cause access privileges to be revoked.

4.6.6 Inconsistencies, Deviations and Rule Violations

A general concern when applying rules, is what rule violations should result in, how and whether violations should be prevented. This also relates to the nature of the rules. In a formal framework for analysing inconsistencies in human-centred systems (consisting of a real-world process and a model mirroring it), two types of inconsistencies were pinpointed [118]:

- *Domain-level inconsistencies*, where a domain rule like "all checks drawn should be registered in the check notebook" is violated, and
- *Environment-level inconsistencies*, where the model no longer mirrors the real process adequately.

Conventional workflow systems *avoid* domain-level inconsistencies. This is achieved by forcing the user to follow a "correct" process, which ensures that all explicit rules are satisfied. Domain-level inconsistencies that are not avoided, will on the other hand be very hard to capture and handle, because they will also lead to an environment-level inconsistency, (unless the domain-inconsistent behaviour is modelled in the workflow). The extra degree of freedom emergent workflow offers the users may cause new domain-level inconsistencies. On the other hand, a flexible workflow system increases the likelihood of capturing the human behaviour that caused the rule violation. It needn't be done "outside the box" of the system, like in a conventional workflow setting. So, inconsistencies that are not avoided by enforcement can be *captured* by the system and then *resolved* by the users or automatically by the system. Because of this, emergent workflow combined with a business rule system might actually serve quality control better than rigid workflow solutions, relying on the skills of workers rather than process standardisation (cf. Table 1 on page 42). Human involvement in the resolution of domain inconsistencies means that advisory rules can be applied, whereas the enforcement paradigm of static workflow only handles absolute rules.

4.7 Activation by the Awareness Engine

In order to coordinate cooperative work, we need to comprehend the actions of others. Comprehension requires that information of what has happened is available. In the CSCW community, workspace awareness has been a topic of research since the early nineties [141]. *Awareness* is defined as *"having perception, knowledge or realisation"* [196]. Dourish and Bellotti [141] distinguish between awareness as a *product* and as a *process*: The process is the continuous cycle of extracting information from the environment, the product the state of understanding about others' interaction with the workspace. Awareness lowers the overhead of working together, creates new opportunities for serendipitous collaboration, and provides people with a larger context for their actions. Awareness information supports the collaborative assembly to achieve *intersubjective comprehension* of the purpose and conditions for the collaboration [30].

Thus, there are two main objectives of awareness services, to help participants to comprehend the collaboration (the *knowledge level*), and to coordinate activities (the *relationship level*) [436]. These levels are closely interrelated. Awareness relates to *articulation work* [32, 439, 468]. Articulation overhead (extra work associated with notifying others of what you are doing) and information overload are emphasised as critical factors for awareness and coordination. Thus, we need to balance information overload with the participants' needs for continuously interpreting the meaning and context of their work. Mediation of the contextual meaning of awareness information is especially challenging. Interactive models are useful for attacking all these challenges. Models provide a context for awareness information, making it easier to filter and interpret it.

Workitems Provide Context for Awareness Information

The worklists in WORKWARE present all items belonging to a project, group, or individual. The worktops of individual workitems gather all the information needed for performing the work. The work management tool is thus a natural candidate for integration with an awareness service, where the awareness information can be presented alongside the other information about the workitem. This remedies some of the problems often associated with awareness mechanisms, e.g. information overload and the lack of context for notification messages.

Collaborative Workitems

If multiple people are assigned to an atomic workitem and they have not found it worthwhile to articulate a detailed plan with separate responsibilities, they need to cooperate closely. The workflow engine does not support such coordination. The awareness engine, on the other hand, keeps the participants informed of what the others do with the shared information, facilitating coordination by mutual adjustment. This support is enhanced when the awareness mechanism also monitors events related to documents and other shared artefacts referenced by the workitems.

Partially Structured Processes

The enactment engine can handle coordination of fully connected workflows. For collections and partially structured workflows, awareness services help the users to control the work. Being notified of the progress and status of other work items, users are informed to make their own decisions about when to do what.

Flexible Resource Allocation

A wide range of techniques has been proposed for flexible allocation of organisational resources to workitems. Some organisations have *job queues* that contain workitems in need of specific personnel resources. Such queues may be split into role queues, based on skill requirements. While this solution allows people to pick ready tasks, it requires maintenance of organisational role models. This approach shares some of the inflexibility of role restrictive systems, e.g. rigidity, vague role definitions and inability to utilise existing skills due to a normative use of organisational roles. A contextual and personalisable awareness mechanism may be a better solution for the personnel allocation problem. The context given by the work process definition, coupled with the possibility for each user to customise which kind of work they are interested in as well as competent for, can be used for filtering which incoming items each user should be informed about.

Coordinating Interdependent Workitems

A WORKWARE process model is perceived as a plan that can be adapted to the situation at hand. One typical exception to the planned sequence of work, involves early start of workitems that according to the process model are not *ready*; i.e. an opportunistic involvement that departs from the model in order to speed up the process. Consider the situation in Figure 48b. Someone has started item B although item A has not signalled that it is ready (it has not activated the flow). The flow will thus enter the state *flow violation* (in Figure 35). In this case, multiple workitems, modelled as sequential, are active in parallel, and need to be coordinated. If the users choose to remove the flow, consistency is restored. But if the flow is not removed, a new situation arises in which it is *re-interpreted*. It is still regarded as a dependency that must be coordinated, but the coordination is supported by the awareness engine, not the enactment engine. Until the flow is activated, the awareness engine should thus inform the actors of A of what happens inside B and vice versa. This means that the flow becomes a *channel for event notifications*.

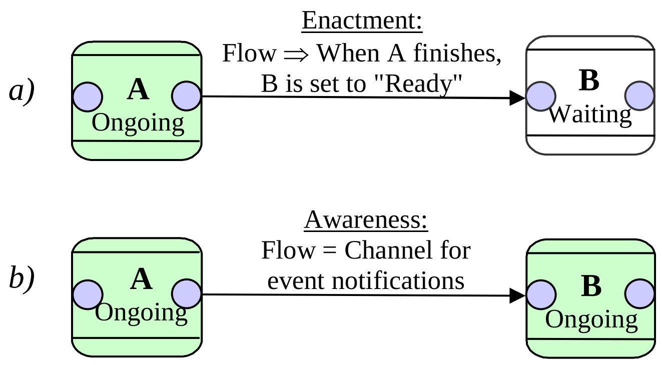

Enactment:
Flow ⇒ When A finishes, B is set to "Ready"
a)
A
Ongoing
B
Waiting
Awareness:
Flow = Channel for event notifications
b)
A
Ongoing
B
Ongoing

Figure 48. Different interpretations of a flow depending on the context.

Collaborative Planning

In an emergent workflow a partial and evolving model articulates knowledge about the work to be performed. During project planning, the awareness engine disseminates articulation events helpful for maintaining a shared understanding. With the coupling of workflow and awareness services, the vision of *collaborative planning* [477] can be extended to cover collaboration in planning, not just divide and conquer where each participant takes responsibility for one part of the overall process. For the workflow participants, notification of changes to either the overall plan or the sub-plans related to their work, is vital for maintaining shared understanding.

4.7.2 Awareness Semantics

A generic awareness model has been applied to facilitate the above services in WORKWARE. The model was developed by Rolfsen [416]. It sees the workspace as consisting of *artefacts* and inhabited by *actors*. There may be *dependency* relationships between artefacts, and actors may *participate* in the processing of artefacts. Events are generated by actors' work on an artefact. All events relevant for an artefact are presented alongside the artefact. For WORKWARE, workitems are treated as artefacts. We also added a customisable event propagation service, where artefact dependencies are channels for events to flow along. These WORKWARE relationships are used as dependencies:

- Flow,
- Is-filled-by, for document events,
- Decomposition.

Participation relationships between actors and artefacts follow responsible, participant, and customer relationships in the process model. The whole awareness workspace model is thus just a view on the existing process model, and no extra articulation is needed. The awareness service is personalised and customised by adding *lenses* [50, 518] to the dependencies. Each lens filters out some of the events while allowing others to pass through. In order to reach a user, an event notification must travel through the process structure without being filtered out by any of the lenses it encounters. The lenses currently defined in the WORKWARE prototype include:

- *Event type lens*, which filters out events of a certain type, e.g. read events, write events, create events or document events.
- *Relative time lens*, which filters out all events older than a given number of days.
- *Absolute time lens*, which stops all events that occurred before a fixed time. This kind of lens is created when the user triggers the *Catch up* service.
- *User lens*, filtering out all events caused by a given user (e.g. myself),
- *Event list lens*, removing all the events that are listed.

Two sibling workitems are related indirectly through their parent. They may also be more strongly related through a flow, and in most filter configurations this stronger relation increases the notification flow between them. This scheme is holistic. Any object or relationship may be part of a path between any other pair of objects, and thus influence the flow of awareness between them. Consequently, the addition or removal of an element may potentially influence all other elements in the model. Since the awareness propagation is based on a general graph structure, it can easily be adapted to other interactive models, similar to the AETHER awareness engine [428].

4.7.3 Awareness Semantics for Flows

WORKWARE's awareness engine utilises the process structure to inform people, in a contextual manner, about the actions of others. In doing so, it provides an interpretation and activation of the flow concept that complements that of the enactment engine. The latter coordinates the flow dependency by *automated sequencing of tasks*, while the awareness engine helps users to *mutually adjust concurrent tasks*. Figure 49 shows the interpretation of a flow by the awareness and enactment engines. The awareness engine handles one typical exception to the basic flow interpretation, that of opportunistic involvement (someone starts work on the receiving end before flows that they were supposed to wait for were activated). The *flow violation* state of Figure 35 is thus replaced by a new state, that of the flow as a channel for awareness notifications. This activation is performed:

- When two tasks connected by a flow are active simultaneously, the people working on either task are informed about what the people in the other tasks have been doing.
- A real *flow violation* now happens only in the case where the receiving task (*Flow.To.Owner*) finishes before it has received its input from the origin task.

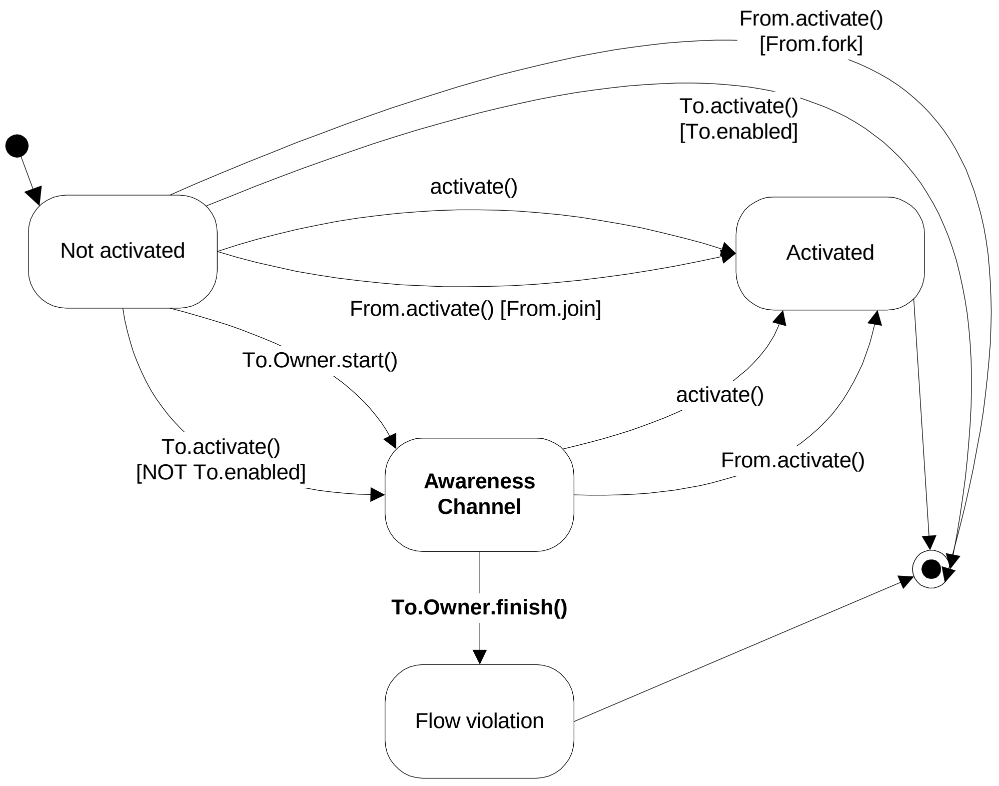


Figure 49. Flow interpreted by the awareness engine.

4.8 Model Driven Access Control

Access control is about deciding and enforcing *who* has access to perform which operations on *what* information. In an emergent workflow system, access control plays two vital roles:

- *Controlling flexibility*, limiting the default full freedom of users to violate the modelled flow of work, to change assignments etc.,
- *Controlling access* to resources, information, tools, and services.

Access control policies in operating systems and repositories are typically defined with system-oriented terms like files, directories, users etc. A typical example, the standard for Distributed Authoring and Versioning on the web (WebDAV [235]), defines access control policies with five main components:

- Who is given access (*subjects*),
- What are they given access to (*objects*),
- What operations are they given access to (*actions*), e.g. read, write, execute, manage access rights,
- Whether access is granted or denied (*negative*),
- Whether the access privilege is also valid for components of the object (*inherited*).

As a component in a process support system, an access controller can utilise the information articulated in the process models. This information includes who works on which workitems (in what type of role), what information and tool resources they use, and how different items of work depend on each other (through decomposition and flows). By utilising this information the access controller can support:

- Access control policies with user-oriented (model) concepts,
- Changes to the model that are automatically reflected in access privileges. For instance, when a person is assigned to a workitem she is also given access to the tools and information needed for performing the work.

The access controller thus becomes a model interactor, activating the process models for a specific purpose that complements the functionality offered by other components. Previous research on access control in workflow environments focuses on static systems [49], and to a limited degree on adaptive systems [68]. Conventional mechanisms were found to be insufficient for dynamic workflow environments [68]. Their major objective is to enforce a strict least privilege policy where users are only allowed to do what their role on the current workitem allows in the current state. In an emergent WMS, other policies, e.g. "need to hide" may also be applied. Where existing approaches rely on static organisational structures, WORKWARE also activates dynamic process structures.

4.8.1 Access Control Policies

The WORKWARE access controller provides a mapping between modelled objects and WebDAV access control policies. In addition to modelled individuals and groups, the subjects recognised by the system include:

- All participants on a workitem,
- The responsible of a workitem,
- The customer of a workitem,

corresponding to the different person role types defined in section 4.2.6. Access rights can also be assigned to a specific role by name, e.g. "Project manager" or "Reviewer". In order to further structure the access control policies, we introduce the concept of a project, which includes every workitem that is part of a single work breakdown structure. The project is thus identified by its root task. The aggregate groups *project participants* and *project customers*, include every person that fills participant or customer roles on any of the workitems in the project. The *project responsible* is the person responsible for the root task (most often the project manager). Since the system is based on an open standard, user groups and units in organisational models may also be used to structure the subjects. The *objects* that users are given access to include:

- Information objects (documents etc.),
- Workitems,
- Projects,
- Information about persons,
- Process models.

For workitems and projects, users can be given specific access rights to certain *aspects* of the objects, including *Description*, *Person roles*, *Information resources*, and *Subitems*. More detailed policies can grant access at the level of single properties, e.g. allowing outsiders to read the name of the document but not the content. The *actions* that subjects may perform on the objects in principle include every tool or service defined in the models. For convenience, actions are grouped into these levels: *Add*, *Read*, *Execute*, *Write*, and *Manage*. *Add* means that new objects can be added but existing objects cannot be read, while *Execute* allows read and the alteration of *state* properties of the objects. *Manage* involves the right to grant others access to the object. With the exception of *Read* and *Add*, these access levels are hierarchical, so that *Execute* implies *Read*, *Manage* implies all other rights etc. The access policies control what the work management tool shows in worklists and worktops. Secondary effects through the enactment engine are however not restricted. When someone finishes their workitem, the next item becomes ready even though the user does not have execute privileges on that item.

4.8.2 Usage Examples

The access controller can be utilised for a number of different purposed. Here are some detailed examples.

Controlled Flexibility

The LAW prototype (cf. section 3.3.4) supports varying degrees of empowerment by attaching access rights to work decompositions [164]. The WORKWARE access controller supports the configuration of LAW in this manner:

- CAN: Workitem responsible (and perhaps the participants) has *write* access.
- STRICT: Workitem performers are denied *write* access, but have *execute* rights.
- MUST: Like STRICT, but *add* is allowed for *sub-items* and *person roles*.

But a wide variety of other schemes can also be supported, e.g.

- Customer controls the workitem model (write access), so responsible must negotiate changes with her.
- Responsible has full control over the item, but is not allowed to delegate it to someone else (denied write access to the responsible role).
- Each workitem can only have one person working on it (the responsible), and nobody is allowed to see the items allocated to other people (deny access to participant roles for everyone, grant read and execute access only for the responsible of each task). This configuration mimics the behaviour of static workflow systems, trying to relieve users of the burden of manual coordination.

The access control scheme is designed to meet all degrees of empowerment, from full autonomy of all users to centralised control with very limited user rights. Policies like "need to know" or "need to hide" can be defined as high level rules and inherited. Such inheritance is implemented as reuse, and will be dealt with in more detail in Chapter 5.

Anonymous Users

In some cases, the identity of who plays a specific role should be hidden to the other participants. In anonymous review processes, the reviewers should not know who the authors are and vice versa.

Virtual Enterprises

Virtual enterprises span organisational boundaries. Although they are involved in a co-operation, partners often want to hide some of the details regarding their ways of working from the others. Defining access control policies based on an organisation model can support this. As the cooperation matures, becomes more formalised, or mutual trust increases, one may find it useful to increase the sharing of information. A dynamic access control infrastructure, where general privileges can be inherited, simplifies this process.

Dynamic Access Control

When the state of a workitem changes, the access rights may also need to change. If the organisation wants to prevent early start of workitems (in general or for some particular items), they can decide that write and execute access is not active until the item becomes ready. Another use of dynamic access policies is to declare that all finished workitems or information objects are locked. Rules that couple the state of the workitem to access control policies enable this functionality. Chapter 5 describes a dynamic classification scheme that supports this.

Visual Representation of Access Rights

APM adds a lock symbol to actions that the users are not allowed to alter, but does not provide a scheme for visualising more complex access rules. Clearly objects that the current user is not allowed to read, should be hidden from the visual process model, but in some contexts it would also be convenient to view and alter the access policies in the model editor. A visual model provides overview of the whole process and is thus a suitable interface for locating unintended security holes etc. Users with manage rights should thus be allowed to view the model as it would appear to other users and user groups. In Figure 50 different access rights are visualised with different colour on the top right icon (red=read, yellow=execute, green=write). We may also attach icons to actor roles to articulate the rights if people filling those roles.

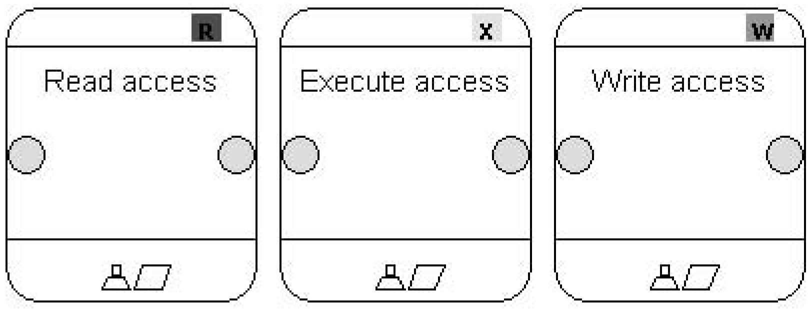


Figure 50. Visualisation of access rights.

4.9 Concluding Remarks

The WORKWARE modelling language combines a number of different perspectives. Striving for simplicity and user-orientedness, we have taken a *transformational* (input-process-output) starting point. APM and WORKWARE also include a rich set of primitives for modelling the resources of each work item. Organisational, material, information and tool resources can be modelled. Further, some aspects of *language action* are integrated (interaction patterns, customer-performer roles), as well as *constraints* (adding details means narrowing the scope of behaviour) and *role* modelling (graphical containment represents areas of authority and responsibility). *System dynamic* perspectives [2] of emergent behaviour, mutual and non-linear causal relations are reflected in the holistic and situated enactment rules. This reuse of tried and tested techniques further support the claim that the interaction framework is feasible and relevant for workflow management. We have also shown in this chapter how the interaction framework can guide systems design. The WORKWARE prototype is one of few current attempts at re-interpreting the workflow concept to include processes with emerging structure. The main contributions are

- To allow ambiguities in the models and to resolve them interactively in the situations that arise, without having to resort to modelling,
- To facilitate dynamic interpretation of model elements,
- To integrate multiple model interactors, complementing the enactment engine.

Though rudimentary in some aspects, the system shows how a number of different perspectives can be integrated in a rather simple language. Mappings between process models, workspace models and directory structures enable generic awareness notification and access control functionality to be utilised without complicating the core language. Reuse of interactor *policies*, for enactment, user interfaces, access control and awareness, is a core challenge for the next chapter.

Chapter 5
Reuse

This second part of the WORKWARE design deals with reuse. *Model reuse* is a fundamental problem often overlooked in languages for IS development. While object-oriented languages support syntax inheritance of *single object* classes, what it means to inherit, specialise and compose *models* (fragments with several objects, relationships and behaviour included) is not that straightforward. In this thesis a pragmatic approach based on user requirements is chosen. Because of the ad-hoc nature of most work, reuse is seen as an *interactive* task, which seldom can be fully automated. The role of inheritance is to simplify articulation by providing default values that users may change later. Emphasising the uniqueness of each process, WORKWARE's metametamodel allows local language extensions and dynamic classification structures. All models are articulated at the instance level.

5.1 Motivation

In Chapter 2 a number of requirements for model reuse were presented. The foundation of our approach was outlined in the lifecycle of process model evolution in section 2.5.7. For this approach to work, end user articulation must be facilitated, and motivated by the usefulness of the activation mechanisms. It is thus impossible to view reuse as independent from articulation and activation. The requirements for articulation and activation must thus be kept in mind also for the reuse framework. Section 2.5 presented a number of specific needs for model reuse, including

- Structured template repositories, facilitating incremental definition, identification and selection of suitable templates,
- Mechanisms for generalisation, comparison, qualitative and quantitative analysis,
- Description of suitable usage context and previous experience for each model,
- Representing and facilitating the reconciliation of conflicting perspectives, and
- Composition of templates as part of ongoing process articulation and adaptation.

The state of the art survey in Chapter 3 showed that although numerous techniques have been proposed to solve each of these problems, a combination that is simple, flexible, and effective for end users could not be found. In particular, most approaches focus on *automated reuse semantics*, rather than making models available as resources for situated adaptation. An interactive reuse approach better matches the contingency of knowledge intensive work. As pointed out by a number of studies [122, 441], knowledge management must be integrated with everyday work to be successful.

In current research, there also seems to be a focus on high-overhead documentation approaches, where models are made especially for the purpose of reuse (*pushed* onto the projects). This approach should be complemented by need-driven mechanisms (*pulled* by the projects). Where conventional wisdom regarding software reuse states that reuse must be planned for, interaction allows need-driven reuse of models that were designed to meet local problems. After all, in most tools commonly used by process participants today, reuse by copy-and-paste is more frequent than structured inheritance, delegation or composition. Our aim in this chapter is to present a reuse framework that integrates these schemes, combining their strengths. As always with interactive models, the degree of automation is an important question. We find that a single solution will not fit all usage situations, and present a customisable framework where the degree of software support can be adapted based on the availability of templates and the knowledge of the process participants.

5.2 Instances, Templates and Classes

In order to support local modifications, fluent articulation, and flexible activation, emergent workflow models represent *instances*. A prototype/instance mechanism is thus applied in WORKWARE. The concept of a *template* is essential. A template is an initial definition of an object instance. A template (prototype) is itself an instance, thus the mechanisms we design for template reuse can also be applied to ordinary models. This results in a simple and general reuse approach.

In fact, the *class* concept was introduced to the WORKWARE metametamodel (Figure 51) purely with the aim to simplify reuse. In this scheme, a *basic class* is just a template with a name. The template may be especially designed and adapted for reuse, but it can also be an ordinary model element *promoted* to the rank of template. The basic constructs of the modelling language (Figure 29 on page 95), e.g. *Workitem*, *Decision*, *Flow*, *Person*, are defined as basic classes. All specification of structure, behaviour and other features, are defined in a uniform way at the instance level. Class features are thus defined in the template. Each instance has a set of properties represented as *attributes*. An attribute is a name-value pair. Attributes and objects (instances) are *typed*, but in order to enable flexible modelling, the typing is advisory. An object may alter its type at any time during its lifecycle. The type of an object refers to a class. Attributes types are *relational*, i.e. refer to another object, or *simple* data values (numbers, texts, dates etc.). Relational attributes are used for encoding the associations of the process modelling language, e.g. *Owner*, *Source*, *Target*, *Customer* and *Responsible*.

This metametamodel may look quite complicated, with a large number of constructs. The end users, however, only encounter a few of these constructs when they define language extensions (through metamodelling). In normal operation, the user only has to care about individual *objects* and *attributes*. Types are needed when a new attribute is defined. Normally, classes are just used as selection criteria for presenting alternative attribute values to users, and for attaching behaviour to a range of objects.

The self-containment of object instances is a core feature of this architecture. Multiple meta-levels is one of the most challenging modelling concepts to grasp for end users, thus a simple, interactive system should apply as few levels as possible. In a WORKWARE object (instance), the data are grouped together with their metadata (attribute definitions). In most infrastructures, e.g. UML [377, 381], Java, and Smalltalk, metadata are stored at a higher level (the class object) than the data they describe (the instance object). With WORKWARE's instance-oriented framework users need not worry about the differences between classes and instances when they are modelling. Classification becomes a method for managing templates in the repository. It is not something that must be determined when models are first constructed. This simplifies local modifications and removes conceptual complexity. The aim of this chapter is to show how

WORKWARE's reuse mechanisms utilise this simple framework, making it also more flexible and powerful than the more complex conventional infrastructures.

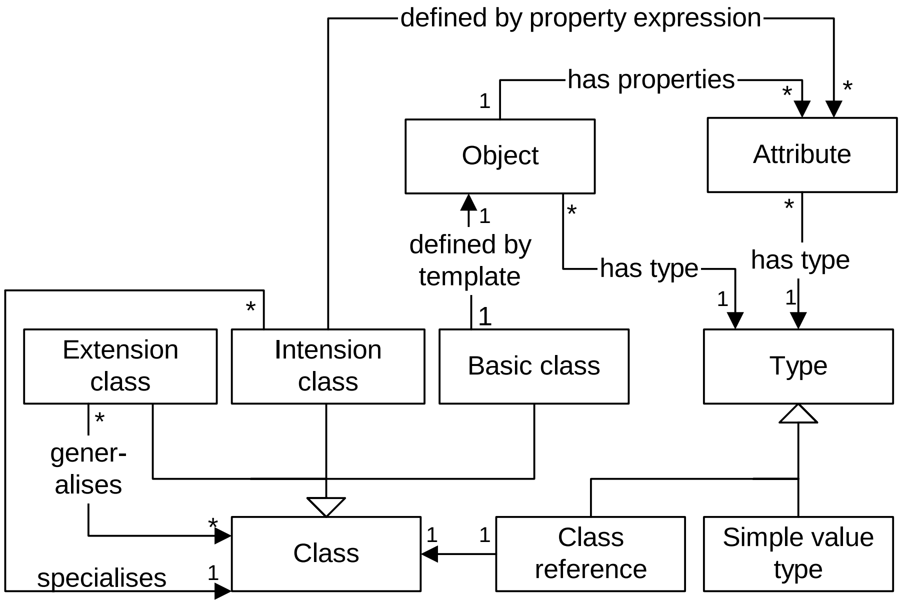


Figure 51. The WORKWARE metametamodel.

5.2.1 Emergent Classification

In the WORKWARE implementation, all instances keep a pointer to its basic class as a *type property*. The type relation is originally inherited from the template during instance creation, but may be altered. In addition to basic classes, generalisations and specialisations can be defined incrementally by their

- *Intension* [81]. Specialisations and property-defined classes [393] are defined by selecting a class and defining the criteria (property values) that members of the new subclass must fulfil.
- *Extension* [81]. A superclass is defined by listing the classes that it generalises.

These schemes were not originally part of the framework. They were added when it became evident that they would meet practical requirements. For instance, classification by intension allows us to separate *fork connectors* from *join connectors* (as discussed in the preceding chapter), while retaining one basic class for all connectors. The separation between join and fork were needed for the implementation of the interactive enactment semantics, as defined in section 4.5. Generalisation by *extension* was added so that common properties and behaviour, e.g. user interface components, could be defined for a range of similar classes.

In pure instance frameworks, classes are not used at all [358, 483]. The conceptually cleanest way of expressing classes [81, 393, 394, 396] would be without reference to other classes. Instead we could define classes by listing the properties required for membership (by intension), and dynamically scan all objects to see if they fulfilled these requirements. Extensional sets could be defined by explicitly listing all their member objects. But this conceptually correct scheme would not be as scalable as the more pragmatic one chosen here. Scanning every object in the system to see if they meet intensional requirements would not be computationally efficient. Aiming to support users in *incrementally* building classification structures, we offer mechanisms that refine the existing hierarchies by adding specialisations and generalisations. This design is thus a direct result of viewing the modelling language as evolving, incomplete and semi-formal.

Extensional generalisation makes it simple to attach behaviour to a wide range of different objects, e.g. all objects which should have event logging enabled. We found it more convenient to do this at the level of basic classes than at the level of instances. The basic classes represent essential features of an object, e.g. it is seldom relevant to change the type of an object from workitem to person. The ability to attach behaviour to intensional subclasses also makes classification structures less rigid, rendering the benefits of the theoretical approach less relevant for customisation.

5.2.2 Adaptive and Emergent Workflow Mutation

As we discussed in section 3.3.2, several approaches to adaptive workflow are based on the assumption that models define workflow classes (schemas), and not instances. Some researchers [45, 213] also claim that interactive modelling by process participants is not feasible, that although ad-hoc changes will occur, they cannot be handled by ordinary users updating the models in an uncontrolled way. This claim is based on the view that workflow class modelling is difficult and time consuming, and requires expert knowledge not possessed by an average user. The WORKWARE approach circumvents these problems, by not hardwiring process definition to the class level. Our basic objective is to support the planning, articulation, coordination and performance of unique workitem instances. We thus enable modelling of instances as well as classes. In doing so, we remove the great complexity of having to look at all past, present and future instances of a class whenever you modify a model. The claimed impossibility of ad-hoc modelling stem from a paradigm that does not separate modelling from class definition. It is not a fundamental problem of all process support technologies. The case studies in Chapter 2 and Chapter 8 support our claim that process modelling by end users is feasible.

In WORKWARE class template changes do not automatically update running instances. This is a logical consequence of treating every instance as unique. We will in the following sections describe *propagation services*, which update instances with generic changes. Such dynamic changes are however interactively controlled, and not a mandatory step. The key feature is that WORKWARE does not require consistency between the instance and its template. They are assumed to be different, the template is a resource for adaptation, a starting point for creating new instances[6]. Figure 52 summarises the difference between WORKWARE's metametamodel and the class-oriented approaches that dominate static and adaptive workflow management systems. It shows that while conventional approaches have a linear relationship from class to instance, interactive approaches supports interplay between the general and the local, facilitating learning and improvement anchored in practice.

[6] Access control may of course prevent users from making local changes, thus enforcing more conventional class-instance relationships (cf. section 4.8).

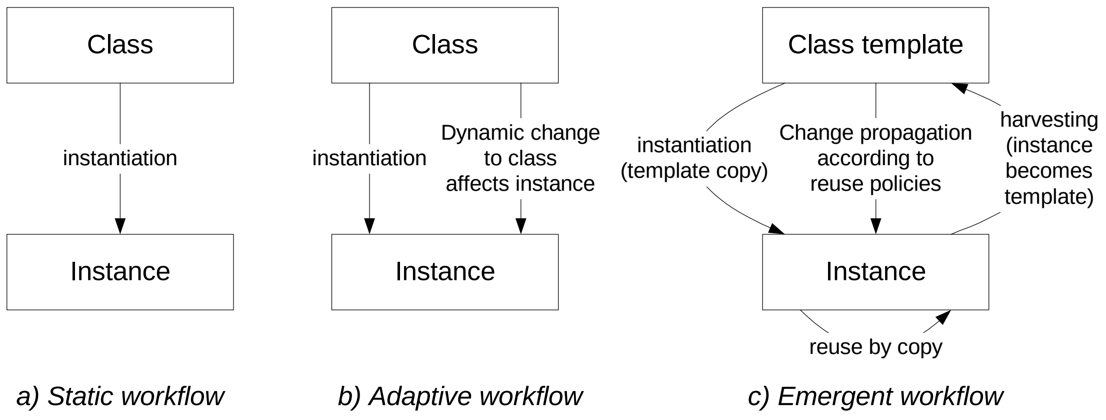

Class
instantiation
Instance
a) Static workflow
Class
instantiation
Dynamic change to class affects instance
Instance
b) Adaptive workflow
Class template
instantiation (template copy)
Change propagation according to reuse policies
harvesting (instance becomes template)
Instance
reuse by copy
c) Emergent workflow

Figure 52. Relationships between classes and instances.

5.3 Inheritance

This section describes how dynamic changes are reused in WORKWARE models, enabling incremental model definition. Inheritance is conventionally defined as reuse coupled to specialisation [483]. For enactment, specialisation relationships played no part, but for reuse, they are crucial. For instance, the PROCESS HANDBOOK uses specialisation in its model repository, both for navigation, selection and incremental definition of templates [329]. This does not imply that specialisation is the only kind of relationship relevant for reuse. The handbook also utilises de- and re-composition [47]. Similarly, software engineers have discovered that reuse mechanisms involving *delegation* can complement inheritance [173]. Access control mechanisms support inheritance along decomposition hierarchies. *Roles* are also powerful reuse structures [149], as well as the history reflected in persons' assignments to tasks. Personal preferences should be reusable along this trace. Thus, our reuse strategies does not rely solely on conventional inheritance, rather it allows reuse along any model structure where it is useful. This represents a holistic approach, where all elements may recursively contribute to each other's definition.

5.3.1 Language Extensions for Reuse

A few extensions must be made to the modelling language in order to support the reuse mechanisms of WORKWARE. These changes are defined in Figure 53. This vocabulary enables us to define and control the semantics of instance creation and dynamic change propagation from templates to instances.

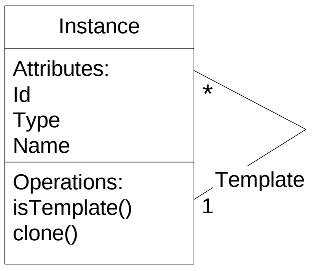


Figure 53. Language extensions for reuse.

The generic attributes for all objects include a unique identifier, a user-oriented *name* and the *type* of the object. Each object also has a connection to its *template*, the object it was originally cloned from. The template is itself an ordinary object. *Template* is a role that every object potentially can be assigned. Figure 53 also shows two new methods, *clone* and *isTemplate*. The latter returns a *true* if this item is declared a *template* for its *type*. The method *clone* generates a copy of the object and any objects whose existence depends on it, e.g. the *input* and *output* decision connectors of a workitem. All workitems, flows, decisions, resources, roles and other process model elements inherit the attributes and associations defined in Figure 53.

5.3.2 Dynamic Change Propagation

So far we have seen how WORKWARE implements classification and instance creation. The final component in our foundation for interactive reuse manages dynamic change. The problem we deal with here is how a change to a general class should affect the objects that already exist when the change occurs. Delegation and lookup schemes are common solutions to this problem. At runtime, requests for inherited features are delegated to the element that defines it, e.g. up the inheritance hierarchy. Such schemes allow incremental definition, where each feature is defined in one place. However, multiple inheritance requires well-defined lookup methods, in case there are conflicting definitions of some features.

Interactive instance models face a different usage environment than conventional OO frameworks. Here general dynamic change occurs far less frequently than local modifications. In order to ensure flexibility, *cancellation inheritance*, where instances and subclasses may remove inherited features, is required [483]. Also, as we have discussed, many dimensions of process models may be utilised for reuse, resulting in massively multiple inheritance. Interactive interpretation of incomplete, semi-formal models requires human involvement in the lookup process. In such environments, delegation and lookup approaches are less suitable. Emphasising simplicity and situatedness, we have based our approach on self-contained instances, where all feature definitions are stored locally. General changes must thus be *propagated* to all the instances they should apply to. Given that the model may be incomplete, human decision making may be required in determining the scope of a dynamic change. Human involvement is often easier at the time of change than at the time of access.

Change propagation is a pre-computation of lookup results. Such pre-computation is efficient if the update rate is lower than the access rate. We expect this to be the case for most general changes to interactive models. However, replication complicates general change. In section 5.5.7 we therefore discuss how delegation and reuse by reference copy rather than object copy can help us decrease replication.

Change propagation is managed by the *Observer* pattern [173] (implemented in the Java standard *PropertyChangeListener* interface [191]). Each object subscribes to notifications about change events in the objects it inherits from. This scheme works recursively through an inheritance hierarchy, because when a template copies changes from its superclass, change notifications are triggered and sent to all subclass observers. Changes should only be propagated when the old value corresponds to the value currently in the observer item (i.e. when the property is not redefined locally). This scheme also supports multiple inheritance, where the latest change wins conflicts unless priority policies are defined. The design is easily generalised, so that any relationship between two objects can be the reason for establishing change propagation inheritance. Below we look at how *reuse policies* can control the propagation of *reusable aspects* along different relations utilising this design.

5.4 Reusable Aspects

Our reuse approach is *modular*, and the modules that can be reused are called *aspects*. An aspect is a group of related features. In WORKWARE models, features are linked to objects as attribute values. We use these terms because they fit well with previous work on aspect-oriented systems [157, 351] and multi-dimensional workflow models [213]. The WORKWARE modelling language contains these aspects related to workitems:

- *Properties*
  The attribute values that describe a workitem or another object.
- *Workflow interface*
  The input and output decision connectors and their input and output flows.
- *Decomposition*
  The work breakdown structure of sub-items and their interconnection network of flows and decision connectors.
- *Resource roles*
  The various objects, tools and persons needed in order to perform the work that the workitem represents.
- *Policies*
  A mechanism is a generic capability of some software component, whereas a policy defines the use of such a capability [1]. In connection with an interactive model, policies are sets of rules or preferences that control the activation of model fragments. Policies are themselves model elements, and should be reusable. Enactment, awareness, access control, user interface, and reuse policies are typical examples.

Although these are the basic aspects as defined in the current language, users may add new aspects not included in this general list, e.g. through metamodelling. These are also made reusable in this framework. New aspects are automatically defined for:

- Each relational attribute (e.g. *Children* which gives the decomposition),
- Each type of relational attributes (e.g. all persons associated with a workitem),
- Each type of policies (e.g. all access control policies).

Aspects are also used in access control (cf. section 4.8). In this scheme there is a hierarchy of aspects, from "everything about this object" by way of sets of related features to individual attributes.

5.5 Reuse Policies

The vocabulary needed for defining flexible reuse strategies is defined in Figure 54. In addition to aspect, *policy* is the primary concept. A policy is a customisation specification for a model interactor. A *reuse policy* customises the reuse tool. It defines which aspects are propagated along which model structure. Each policy applies to one object, but as policies themselves are reusable, they can be propagated to other items according to the reuse policies for reuse policies.

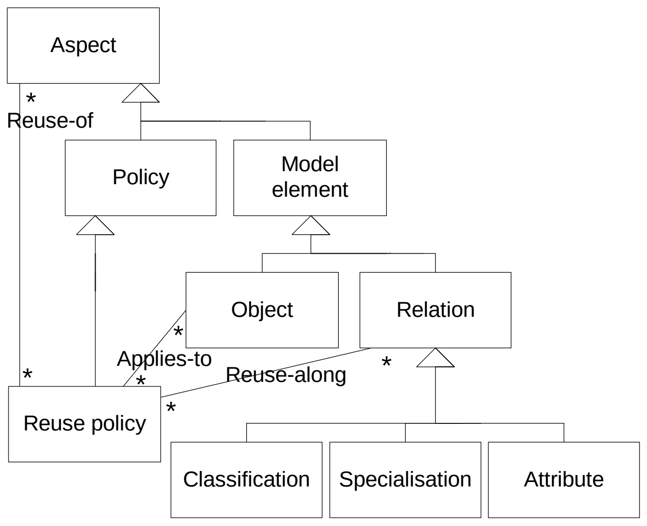


Figure 54. Constructs for flexible model reuse.

A reuse policy specifies the object it applies to, the relation along which it propagates, and the aspect that is reused along this link. Policy attributes control the priority of this policy compared to others; and whether the aspect is cloned, or just the reference to it (deep or shallow copy). The framework allows both instances and classes to be the object of reuse policies. Reuse policies for basic classes are attached to the template, while extensional and intensional classes are defined as metaobjects. Specialisation is a relation between classes. It is automatically computed from class definitions.

In WORKWARE models, reuse among workitem objects and classes are most important. For workitems, classification, specialisation, decomposition (attribute *children*), flows (attributes *in* and *out*), and resource roles (attributes of type *person*, *tool* or *document*) are the most important relations along which reuse occurs. In addition to workitem attributes, decomposition and interfaces, policies for enactment, access control, user interface customisation and awareness propagation, are often reused. In the following subsections, we look at which aspect should be reused along which relations. The discussion is guided by reuse rules in existing process support environments, and by requirements uncovered during use of WORKWARE's modelling language. The set of workitem relations relevant for reuse is summarised in Figure 55.

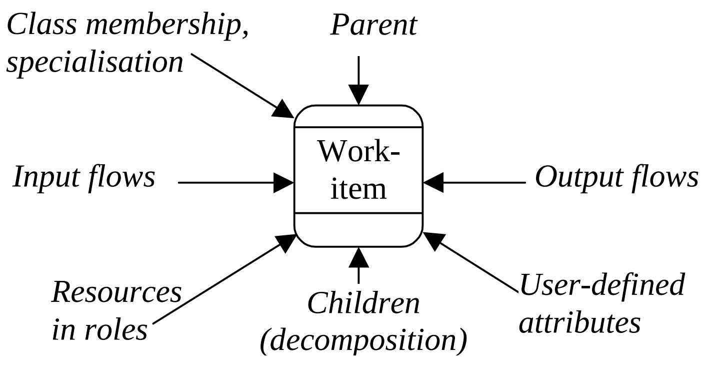


Figure 55. Inheritance relations for a single workitem.

5.5.1 Using Classification and Specialisation for Reuse

As described is section 3.4.3, specialisation inheritance of process models is an established reuse method. Many different specialisation schemes have been proposed, utilising e.g. object oriented constructs to represent dynamic behaviour. For transformational models, generalisations are either unions or intersections of the sets of behaviour of the subclasses [540, 550]. In these examples, specialisation is typically utilised for reuse of process models (decompositions), but the hierarchy clearly has a role to play in the reuse other aspects as well. Workitems of a particular class, e.g. "Write presentation", can be allocated a specific user interface configuration that emphasise the particular needs of such tasks. Default access rights to the presentation file can also be defined at the class level. Enactment policies and components in the model of the workitem, e.g. that the item has an invoked software tool, would typically be shared among items of the class "Write something", and be further specialised for "Write presentation" items. On a higher level, different types of projects require different access and enactment policies, e.g. there may be special rules that apply to projects for specific customers.

In order to facilitate recursive inheritance, all classes must be configured to reuse reuse-policies along specialisation relations. Due to reflection the users only have to specify this for the top node of the inheritance tree, and it is automatically reused by all subclasses. Multiple inheritance is allowed, and managed by reuse policies, which allow users to define what aspects are inherited through which specialisation relation. A "Write presentation" item that is also a "Top secret" item could thus inherit user interface and enactment policies from the first and access control policies from the latter. Priority attributes order reuse policies in case of multiple inheritance conflicts.

5.5.2 Decomposition Structures for Reuse

As discussed above, standard access control mechanisms utilise inheritance along decomposition structures. When a user is given access to a directory in a file system, she is also by default given access to all the files inside that directory. Similarly, we can utilise decomposition inheritance to define enactment policies for a project, and through reuse along recursive children links make them apply to all the workitems in the project (unless overridden by local policies). In addition to enabling general policies for projects or work packages, decomposition inheritance has a role in resource management. Resource decompositions, e.g. organisations into departments and information objects into parts, are suitable structures for reusing access control policies. Although reuse *down* decomposition hierarchies seems most important, there are also cases where the whole should reuse features added to a part. One relevant policy is that the resource signature of a parent workitem should include all the roles of its sub-items. This can be supported by bottom-up reuse along parent links.

5.5.3 Resource Assignment for Reuse

WORKWARE's reuse policies allow user communities to deal with trade-offs and integration of personal preferences with group, organisational, and process-specific configurations. Since customised policies can be reused, adapted and distributed among users, this framework can also support the social processes of tailoring [493, 363], the spreading of local innovations throughout formal and informal networks.

By utilising *is-filled-by* links for reuse, customisations attached to person objects in the model can be applied to all workitems that the individual participates in. When multiple people with conflicting preferences cooperate on larger tasks, the reuse policies allow users to articulate the result of negotiations regarding which preferences and policies should be applied in the joint work. Users may articulate their own preferred ways of working as personal process templates or enactment policies (e.g. "I normally treat deadlines as something that requires a response when they are not met, so the system should notify me when one of my workitems is delayed"). The primary motivation for articulating these policies would be to get more useful support from the system, but the fact that such assumptions are made explicit also enable other users to understand your expectations towards the collaboration.

Skill-dependent procedures allow organisations to benefit from personalisation. Recognising that experts and novices have different needs, different process models can be selected based on who is allocated to a workitem. While novices may need a detailed description of what to do, experts could be given more freedom and subjected to less monitoring. For harder problems, novices may be required to consult an expert before they start the work.

5.5.4 Reuse along the Flow of Work

Flows control the sequence of work (enactment), represent dependencies that should be coordinated (awareness), and may carry information or other resources (shared workspaces and resource brokers). An interesting potential for reuse along flows is the *context-preserving work sequence*. Here, instead of modelling separate resource signatures to each workitem, roles are propagated along the flows, so that all the steps apply the same resources. Changes made to the signature during the performance of one workitem are immediately propagated to the next, including added documents etc. Such a policy simplifies articulation in cases where tight collaboration is needed in a structured process and one wants a familiar context throughout the process. In our case studies we have encountered this need in administrative procedures associated with project proposals.

5.5.5 Property-Controlled Policies

Every property assigned to an object in a WORKWARE model can be utilised to control reuse. Classification by intension defines classes whose members are all objects that possess a set of properties. This enables us to attach e.g. enactment policies to properties. Properties denoting vagueness and uncertainty (e.g. "I am not quite sure about this" or "We will add more here later") should make the enactment engine apply more human involvement than it otherwise would (e.g. for items with a property saying "this is fully specified"). Likewise, a workitem labelled "urgent" should apply different rules than other items. Section 4.6 contains a number of examples of enactment policies that are associated with workitems through properties, e.g. *Autostart* and *Repetitive*.

5.5.6 Mapping and Parameterised Reuse

The reuse framework defined in Figure 54 does not say much about the way reused aspects are applied in the descendant. By default, a direct mapping is assumed, i.e. that the aspects appear in the same way in the target and the source of the reuse link. There are however circumstances where this rule does not meet the requirements. In order to make

WORKWARE behave like ACTION workflow [339] when it comes to person roles in decompositions, we can define a reuse policy that makes the person who is responsible for the parent workitem the customer of the sub-item, i.e.

Self.Customer = Parent.Responsible

This kind of reuse policy requires a *mapping* definition, which connects the reused feature of the source to another feature of the target. Such parameterisation is especially relevant for reuse of property values. For instance, if a generic project model contains a workitem for writing a report about the project, its name can be specified like this:

Self.Name = "Write experience report on " + Parent.Name

As we saw in the previous chapter, the UML Object Constraint Language [517] is suitable for expressing such interdependencies.

5.5.7 Delegation by Reference Copy

WORKWARE represents relationships as attributes pointing to other objects. When inheriting relationships, we are faced with a choice between reference and object copy. Dependent objects, such as the resource roles, input and output decisions of a workitem should be copied alongside the workitem that contains them. Independent objects such as interactor policies and actual resources filling roles, should however not be replicated. Our framework supports both these solutions, and lets the users decide which one to use in each context. Although general rules can be defined and reused, there are cases where the local context influences which method should be applied. For instance, while some cases require a recursive deep copy of a process model, others just need a shallow copy of the top-level process.

Interactor policies customise the behaviour of model elements without modifying the elements themselves. The architecture thus separates the syntax from the semantics. By delegating the control of semantics to policy objects, and by dynamically and explicitly articulating the binding between model data and behaviour, flexibility is ensured. By copying references to interactor policies rather than replicating the policy objects, WORKWARE increases the manageability of system configurations. Reuse policies are themselves model objects, so attribute values may be inherited among them. For instance a default rule could be defined that reuse policies for policies (an intensional subclass of reuse policies) should use shallow reference copy.

5.5.8 The Combination Problem of Multi-Dimensional Inheritance

Multiple inheritance is problematic because the inheritance structure can become complex and unmanageable. This is called spaghetti inheritance. In software engineering, aspect oriented programming (AOP) aims to manage code complexity, reuse and evolution by structuring code based on separation of concerns [157, 351]. Each concern is implemented by an orthogonal *aspect* that *crosscut* class hierarchies, implementing similar functionality for a number of classes. Aspects thus constitute a structure for reuse independent of specialisation hierarchies or other *dominant decomposition* structures for code. In programming, typical aspects deal with security, memory management, resource sharing, dynamic business rules, error and exception handling [351]. In WORKWARE, each interactor implements one aspect.

Aspect orientation reflects the fact that most software problems are multi-dimensional and holistic. From the principle of requisite variety [357], solutions and model structures should thus also be multi-dimensional. Rather than attempting to bring the order and structure of a software program to social organisations, we therefore design systems and models that reflect the complexity of their environment. This approach demands interaction, both in articulation and activation of models. The closed system problem of automatically ensuring that inheritance always works correctly and completely, becomes less important when users are involved in model interpretation and can correct false assumptions made by the software. To develop tools that facilitate user understanding and control of process model inheritance, is thus a core problem, and the topic for the rest of this chapter.

5.6 The Process Knowledge Management Interactor

WORKWARE's reuse scheme addresses the challenges of a wide variety of work processes, roles, and users with different knowledge and experience. Ordinary end users most concerned with getting their work done will probably prefer reuse by copying a template or model fragment (typically a workitem and its decomposition). For many, the additional benefits of propagation after instantiation complicate more than it helps. Project managers are given the opportunity to define local standards for repetitive tasks, like administration and reporting. They may also utilise some of the reuse features to simplify the modelling of project tasks, e.g. sharing of resource signatures between workitems. Process experts and analysts can model company standard procedures and best practices, and utilise the propagation features to make sure that all instances follow the updated procedure. The width of this usage spectrum is depicted in Figure 56.

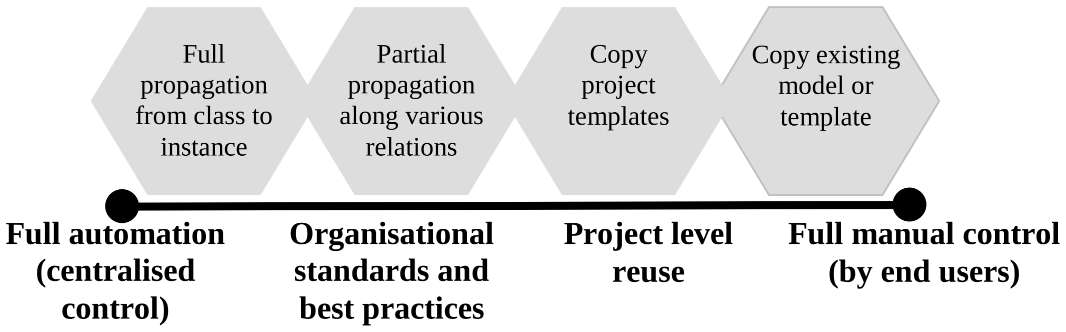

Full propagation from class to instance
Partial propagation along various relations
Copy project templates
Copy existing model or template
Full automation (centralised control)
Organisational standards and best practices
Project level reuse
Full manual control (by end users)

Figure 56. The model reuse tool supporting varying degrees of centralisation.

5.6.1 User Interaction in Reuse

Though the reuse framework is not primarily intended for novice users, project participants may encounter it in some situations:

- When defining a new workitem, users select among a set of available templates. They may also choose to simply copy an existing object.
- Users may harvest their own personal templates, through a "Save as template" service.
- Default values are provided for new workitems according to the defined reuse policies and the context the item is placed in. For instance you are by default responsible

for new items you create, and the responsible of the parent item is by default customer of the new item.
- Project policies for enactment, access control, and user interfaces are automatically applied to new items.

A reuse service can ask users whether change propagation should occur, and add policies when the user selects the *"Don't ask me about this again"* option. Knowledge managers, process experts, and designated super users will more often customise reuse. The provision of template reuse policies for typical scenarios would simplify their job, especially in the early phases. In the following subsections we will explore different means of making reuse customisation more straightforward. Reuse is followed by appropriation of the template into the new environment through adaptation and extension. Hence for an interactive model, the aim of reuse is not to provide a complete solution, but to simplify the articulation work by providing useful starting points.

5.6.2 Organisational Policies for Process Knowledge Management

Organisational control and knowledge management are central reasons for process model reuse. By combining reuse from class to instance with restrictive access control, organisations can customise WORKWARE to function like a conventional WMS. The core benefits of WORKWARE however lie in its ability to support more organic and decentralised organisational structures. It allows organisations to e.g.
- Assign different enactment, access control and user interface policies to different types of tasks and projects (through classification, specialisation and properties).
- Assign different rules for different employees, allowing experienced workers a lot of leeway while requiring personnel in training to follow more elaborate procedures.

Reuse of reuse policies is a key mechanism in the support of organisational policies. This allows the organisation to define global defaults.

5.6.3 Processes of Reuse

One way of attacking the multitude of challenges involved in process model reuse is to look at the metaprocesses involved. The process model lifecycle presented in section 2.5.7 provides the framework for an investigation of reuse processes. The steps (workitems) involved in reuse are:
- Creation of a new model element based on an existing object, possibly including the identification of candidate templates and selection among them,
- Change propagation according to reuse policies,
- Template management,
- Metamodelling, defining the vocabulary for articulation, and
- Harvesting of local innovations, which is the topic of the next subsection.

Model Element Creation

When a model element, e.g. a workitem, is created, it copies default properties, components and policies from its template. It also becomes member of the same classes as its template. By adaptation (alteration of properties and policies), the new object becomes customised. During adaptation new reuse policies may be activated, e.g. when a workitem is placed in a work breakdown structure, it may inherit features from its new parent. Default reuse links may also be broken, e.g. the alteration of a property can remove the object from intensionally defined classes. This implies that the users can do a great deal of customisation simply by articulating the properties of the new object, without caring about the implementation details that follows.

Change Propagation

Based on the principle that all objects are unique and self-contained, the default interpretation (with no reuse policies) is that aspects are copied during creation, but not shared throughout the lifetime of the objects. If a reuse policy is defined, aspects are propagated along the link that the policy applies to as long as the aspect has not been changed in the dependent object. Users may also explicitly declare that a particular change they made should be propagated along some relation. For instance when a user adds a property to an object, she can choose among a number of different scopes for the change:

- Only this object (this is the default),
- All objects of the class that this object is template for,
- All objects of one of the classes that this object is a member of,
- All components of this object (through decomposition),
- All objects that this one is part of,
- All objects that the current user has a role in, e.g. "all my workitems",
- This object and all that follows (or precedes) it in the flow of work.

Through user involvement we are thus able to customise change propagation in each individual case.

Template Management

The reuse framework provides a number of means for maintaining and structuring large template repositories. Reuse links (classification, specialisation, (de)composition, resource allocation, and work flow) act as structuring mechanisms for the set of templates. Personal templates should also be separated from group, project and organisational ones. We expect more extensive use of change propagation in template repositories than in local models.

Metamodelling

Metamodelling implies addition, removal or changes to the set of available language primitives. Classes constitute the vocabulary for process model articulation. The framework presented here facilitates incremental metamodelling in that users can define new classes based on existing ones by simply copying a template and making the necessary changes. Since templates are themselves instances, this corresponds to "metamodelling by example", where metamodelling uses the same means as ordinary modelling. When users want to attach behaviour to groups of objects that have not yet been defined as a class, they may utilise the mechanisms for specialisation by intension (properties) or generalisation by extension (by explicitly selecting subclasses from the set of available classes). As these constructs relate classes to each other in a specialisation hierarchy, changes to some aspect of one class should in most cases be automatically propagated to its subclasses. Metamodelling may also take place locally in an instance model, to meet situated requirements. If these changes are of a general or lasting nature, they can be harvested and included in the template repository.

5.6.4 Harvesting for Reuse

The processes involved in harvesting include (cf. Figure 8 on page 44):

- Generalise an instance model into a template, reset its dynamic properties (state) etc.,
- Analyse, evaluate, and classify templates, in relation to each other, and
- Trace and utilise model evolution histories to gain a deeper understanding.

Like the reuse processes discussed above, all steps in harvesting can be carried out interactively.

Instant Generalisation

The transformation of an instance into a template can often only be partially automated. The enactment metamodels presented in section 4.5 shows the initial states of the different model elements; hence it is straightforward to reset the state of a model fragment by resetting each of its components. Other aspects are more difficult to manage:

- *Resources*, in which cases should allocation be removed, and in which cases should it be preserved in the template? This depends on the scope of the template. If it is a personal one, the user may want to keep herself as the default responsible, but remove the other participants. For project level templates, allocation of workitem roles to project roles may be kept, but allocation of individual performers could be removed. For organisational templates the basic assumption may be that all roles should be made vacant. Similarly, information resources that are used as input should in most cases be kept as part of the template, while information objects that have been produced in the item should be replaced by a vacant role.
- *Granularity*, should the template use shallow or deep copy? Are we talking about a single workitem as a template, or also its decomposition? The decomposition hierarchy structures large process models into fragments suitable for reuse.
- *Parameterisation*, where should actual values in a given object instance be replaced with parameterisation rules? Can we utilise natural language processing to deduce connections between property values?

From these examples we see that full automation of instance generalisation is seldom feasible. Instead, the services mentioned above should be offered in the user interface for harvesting. One should also note that instant generalisation is needed for reuse-by-copy as well. For such reuse the default solution should be simple for ordinary users.

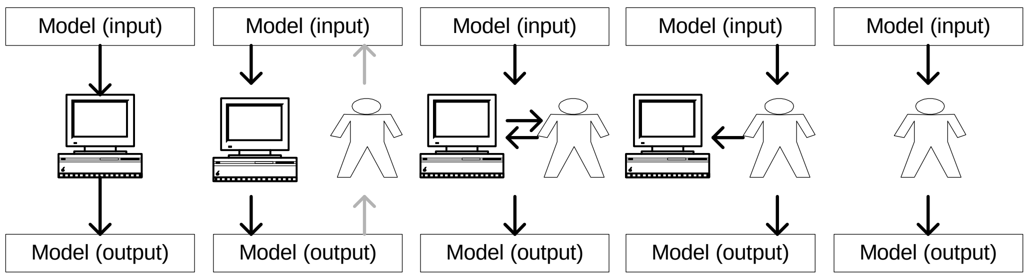
Evolution Traces

The history of an instance model contains a wealth of input to harvesting, e.g.

- Which template was this model (and its components) originally based on. This is input to the classification process, e.g. a model that is based on another could by default be regarded as a specialisation.
- What additions, removals and changes were made as part of the local articulation of the work? This information is useful for describing the specific aspects that are unique to the new template.

5.6.5 Towards Interactive Model Transformation

More fundamentally harvesting and reuse can be seen as model transformations [375]. It is acknowledged that model transformations can seldom be fully automated [421]. The interaction perspective is a suitable framework for integrating transformation techniques. Interaction bridges the gap between fully automated and fully manual transformation. How this perspective applies to the problem of model transformation is illustrated in Figure 57.



Automatic transformation with complete mapping rules | Flexible automation | Interaction: Both automated and manual mapping steps | Explicit mapping at the instance level | Manual transformation with implicit mapping

Figure 57. Spectrum from fully automated to fully manual transformation.

The figure shows varying degrees of transformation automation, from full to none. In between we find various forms of interactive transformation, where some support is offered by software tools but the user still influences mapping decisions. For reuse the input model is the template and the output model is the instance model contextualised to fit the local environment. For harvesting the input is the original instance model and the output is the new or updated template. Fully manual transformation means that the tool offers no support for capturing mappings between model elements in different views. A very simple automated support is thus to add the explicit capture of such relationships, enabling reuse when the input is altered. The simplest form of interactive transformation is to define rules for typical mappings at the class level, e.g. "Each use case is modelled by a number of interaction sequence diagrams". The next degree of automation is to add services that automatically perform some of the mappings from one diagram to another, but still rely on manual transformations of other model fragments. Users may also override the default mappings provided by the tool. The final form of interaction involves a tool that controls the transformation, resolving incompleteness by asking its users. *Flexible automation* means that the transformation is fully automated, but the users are offered a rich set of modelling constructs (properties, parameters or stereotypes) that they can use to control how the mappings are made. By inspecting the outcome, users can then adjust the input model so that they get what they want [375]. The practical utility of this framework includes:

- Integration of a variety of perspectives on model transformation, illustrating how they complement each other.
- Ability to handle transformations of incomplete models with incomplete mapping rules through human involvement in the transformation process.
- Discussing different requirements for different applications of model transformation, ranging from formal specification (full automation) to simple, informal flexibility (manual transformation). Trade-offs between simplicity and automation are central.
- Discussing different transformation techniques according to their scope in the interaction spectrum, and pointing to directions for further development and adaptation of existing techniques to increase their scope of use.

5.6.6 Utilising the Enactment Engine to Support Reuse Metaprocesses

We will now look at how the processes of reuse are integrated. Since reuse and harvesting involves a number of interactively controlled steps, they can be articulated as process templates (Figure 58 and Figure 59), reused, adapted and activated.

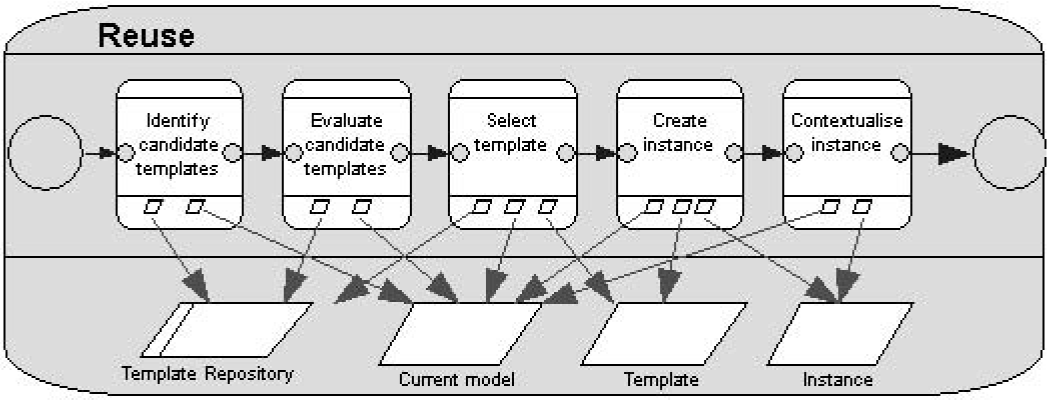


Figure 58. Process model of the reuse work.

The models depict workitems and information resource roles (templates, instances and the template repository). When instantiated, these models will also become populated with persons. They only provide an overview of the steps involved in typical reuse and harvesting processes, and more detailed decomposition or specialised tool support (services for each step) are needed. The metaprocess approach does however yield a number of benefits:

- The models guide novice users, taking them through the necessary steps,
- Reuse and harvesting are here modelled just like ordinary workitems, i.e. there is no conceptual gap between the metaprocesses and the primary processes,
- Metaprocesses are enacted with the same flexibility as normal processes; they can be customised to the problem at hand etc.

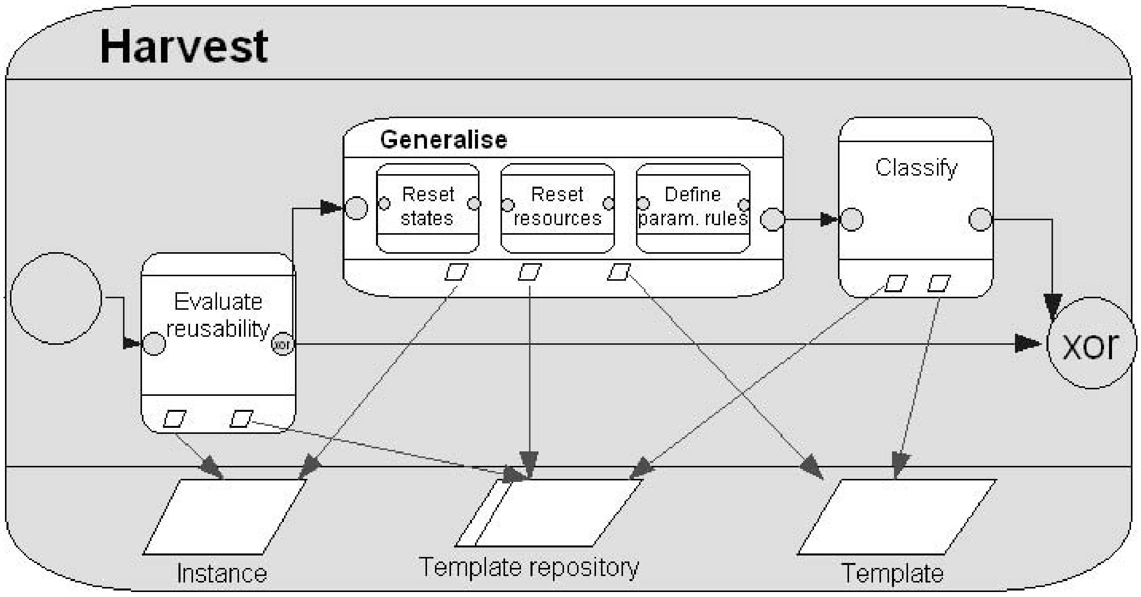


Figure 59. Process model of the harvesting work.

5.6.7 Making Reuse Decisions

In Figure 60 we see a specialisation hierarchy for different kinds of research project templates. Specialisation relations are articulated as XOR *decisions*. This implies that for each step down the hierarchy the users must select one of the alternatives. When a leaf node is reached, or when the user does not find any of the available specialisations useful, a template is selected for the project at hand. Depending on the template, the users may in the next step select templates for the sub-items of the overall template. In this way we model template browsing in the PROCESS HANDBOOK [329] as a series of selection decisions. Specific tools, e.g. for discussions, can be made available to support this selection process. Since the decisions are captured in the trace of the reuse process, we also have a starting point for classifying the resulting model instance, should the users want to add that to a template repository later. In this way WORKWARE is able to support a growing repository while minimising the extra effort required by end users to do classification etc.

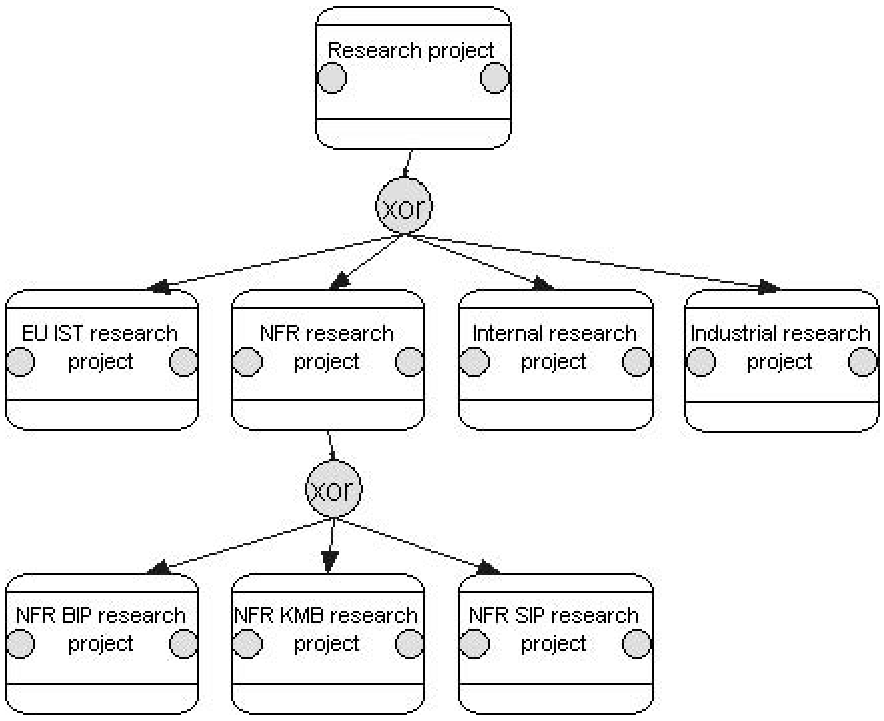


Figure 60. Template specialisation hierarchy as reuse selection decisions.

5.6.8 Summary

This chapter has described the design of an interactive tool for process model reuse, aimed at integrating simple services for novices and more sophisticated support for designated PKM groups and experts. The main features are:

- Simple, instance-oriented metametamodel,
- Mechanisms for incremental classification (by intension or extension),
- A customisable inheritance framework, which work along any modelled link, and
- Metaprocesses for reuse and harvesting supported just like ordinary work.

More details about the use of these techniques will be presented in Chapters 6, 7 and 8.

Chapter 6
Interactive Modelling Techniques

This chapter generalises the designs presented earlier. It identifies techniques for interactive modelling that are relevant outside the particular area of process models. Instance modelling, interactive activation, semantic holism and explicit representation of decisions are highlighted. As discussed in Chapter 2, active and interactive models have been identified as a major direction for future research [102]. In addition to work process support, a wide variety of interactive models are appearing in the research literature:

- Product models for cooperation support [86, 163].
- Conceptual models for database and document management and retrieval [70, 301].
- Enterprise and process models for document management [365].
- Dynamic ontologies for deduction and inference [260].
- Multimedia classification and retrieval (image, video and speech) [23, 511].
- Social network models for making and managing contacts [364].
- Organisational structure models, e.g. for access control [68].
- Conversations visualised in order to gain overview and understanding [136].

These examples represent separate areas of research, but the approach of interaction around evolving, incomplete models creates several common concerns. Although the main objective of this thesis is to show how interactive process models can support knowledge intensive project work, the conceptualisation and most of the requirements presented in Chapter 2 apply to other types of interactive models as well.

6.1 Instance Modelling for Local Modifications

WORKWARE supports local modifications by *instance modelling* [254, 396]. This scheme limits the scope of a change in the first place to the local situation, removing much of the complexity that has prevented modelling by end users at the class level. Instance modelling is a prerequisite for establishing an immediate connection between the interactive model and the domain that it represents. As shown by Parsons [393, 396], a modelling framework based on instances removes problems in schema evolution, integration and interoperability. The framework presented in Chapter 5 complements Parsons' with incremental classification and dynamic change propagation, thus preserving many of the benefits of class-oriented approaches. There is nothing in this design that is specific to process models. Reuse policies that control the sharing of aspects along modelled associations, can be applied to any structured model. As pointed out above, this scheme is a generalisation of conventional inheritance mechanisms, which are applied in a wide range of modelling languages. Here are some examples of situations where the extended functionality of the reuse framework can be useful:

- In an organisational model, rules like "Every manager has access to all that his subordinates have access to" can be operationalised by reuse up the reporting hierarchy.

- In a product model, a relationship like can-be-replaced-by between different components should be *transitive* (but not always symmetric). Transitive relations are reused along themselves, e.g. if A can-be-replaced-by B and B can-be-replaced-by C, the latter relationship is reused along the first so that A can-be-replaced-by C as well. However, relations are often contextual, i.e. although A can-be-replaced-by B in one product, the relation may not hold for all other products. Semantic holism and the priority attributes of reuse policies allow these problems to be captured.
- In a concept model or ontology, synonyms and homonyms are typical relationships that could be operationalised for reuse. If a document contains a term and the user is searching for a synonym of that term, the document may be included in the search result if reuse along the synonym relation is enabled. The opposite is true for documents that contain the search term but where the meaning of the term is different (homonyms).

It can be argued that mass production of products and services does not require the level of local flexibility offered by instance modelling. On the other hand, there are still a large number of cases where such flexibility is needed, e.g. during product design. In manufacturing industries *mass customisation* involves assembly of different components for each product instance. While it is possible to represent the wide variety of customisation options in a class hierarchy with multiple inheritance, the size of such a model would make it difficult to maintain. The number of leaf classes would equal the *product* of number of options for each customisable feature. Instance modelling with a predefined set of available properties and components would only require the *sum* of customisation options to be represented as property primitives. A framework based on object instances and properties thus simplifies such languages.

6.1.1 Exception Handling

Interactive instance models enable exceptions to be built in right up to the time when the models are activated. Compared to completely prescribed models, a greater number of exceptions can be captured and managed, due to the learning of the participants. Capturing exceptions are important for knowledge management, as it increases the accuracy of the model. Instance modelling facilitate exception handling for all the perspectives outlined in Strong and Miller's general overview [464] (section 2.3.4). Exceptions as *unpredictable, random events* create a need for human interpretation of models in the situation that arise. Unpredictable events occur at the time of activation, hence close integration of articulation and activation is needed. By overriding the expected behaviour of the models, users contribute to articulation of unpredicted events. One example is when someone decides to start a workitem early when faced with delays in the preceding items. Bernstein [45] shows examples where reuse mechanisms are utilised to find alternative solutions in the face of exceptions, e.g. a strike among airport workers forces the project participants to look for other means of transportation. Here exceptions are *normal* parts of process flexibility, to some degree anticipated in the predefined model templates. Another example is when new items (process and/or product components) are added dynamically, e.g. because of changing user requirements in a software engineering project. These three examples illustrate three different degrees of exception handling facilitated by interactive models:

- Simple exceptions like rescheduling are handled by manual *activation*,
- Exceptions that have more far-reaching effects may lead to ad-hoc *articulation*,

- Templates provide alternative solutions. By allowing any model to be *reused*, the set of alternatives includes all previously developed solutions, not just those recognised and promoted by organisational knowledge management.

The final perspective, exceptions as *errors*, represents situations where the template should be modified based on experiences captured in an instance model. This is handled by the harvesting process outlined in section 5.6.4. Unpredictable events, normal variation and errors are different perspectives on exceptions, they do not constitute a taxonomy that one can classify events according to. It is often difficult to separate errors from local, unpredictable events, and variation just refers to events for which we have established exception handling mechanisms. From the perspective of interactive models, we separate exceptions into these categories:

1. Exceptions that were not articulated prior to their occurrence, but are tolerated by the activation mechanisms (model interactors) without changes to the local model,
2. Exceptions that are articulated in the local model as alternative solutions (e.g. alternative workflow paths or product configurations),
3. Exceptions that are not included in the local model but where templates exists for their handling,
4. Exceptions that were not articulated prior to their occurrence, and require changes to the local model, and
5. Errors that cause changes to one or more templates.

These categories are ordered according to expected frequency of occurrence. Figure 61 depicts exception levels 2 (to the left) and 3. In both cases item A is followed by either (XOR) B1 or B2, but in model 1 this selection is modelled directly as part of the local process, while in model 2 the selection is modelled as alternative templates for item B (similar to the specialisation hierarchy in Figure 60).

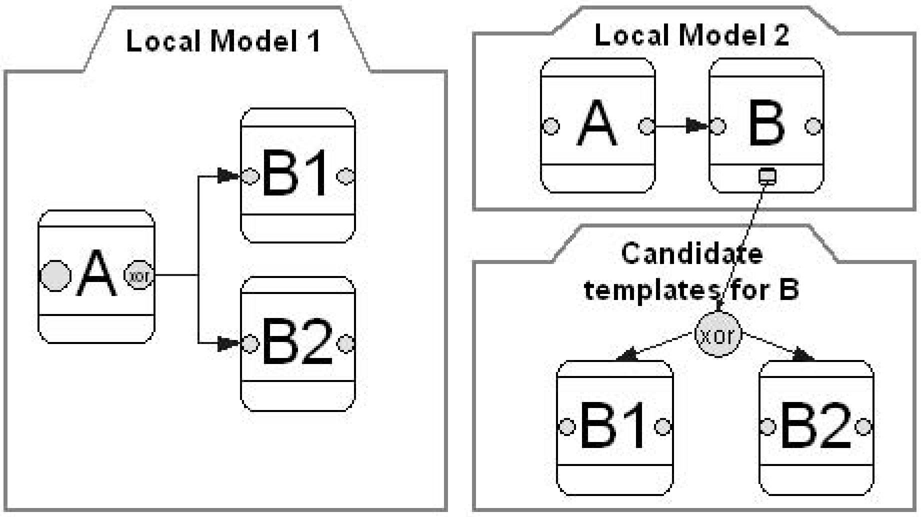


Figure 61. Equivalent models for exception levels 2 and 3.

Both models have their strengths and weaknesses. Model 1 shows the two alternatives as a straightforward decision, while model 2 appears simpler, especially if the selection can be automated. However, the main point is that the language allows users to handle these exceptions is a uniform way, by *making decisions*. Reuse (template selection) and activation (workflow routing) is integrated. Decision-making will be elaborated in the next section. This discussion is based on a description of exception handling in information systems in general. Exceptions may occur more frequently in process and product models than in e.g. concept, social network or organisation models, but from the discussion in Chapter 2 we can argue that models immersed in day-to-day work practice are always faced with exceptions.

6.2 Explicit Representation of Decisions

Explicit representation of decisions regarding the flow of work was one of the few adjustments made to the APM language in order to make it suitable for interactive enactment. In the previous chapter we also saw that specialisation hierarchies for process templates reflect reuse decisions. In this section we explore other cases where this technique facilitates interactive activation. We thus propose decision modelling as a general technique for converting conventional languages into interactive ones.

6.2.1 Reuse Decisions

Figure 60 shows reuse decisions among alternative templates in a specialisation hierarchy. The relation between decisions and templates in this hierarchy is called *candidate*, since the various templates are candidates for filling a role in the local instance model. In the specialisation hierarchy the decisions were modelled as XOR-decisions, because the users have to select one of the candidates. Along the decomposition relations (work breakdown structure), on the other hand, we find AND-decisions: When you select a template, you select a workitem decomposition, including all of its components. On the next level, for each sub-item, alternative candidate templates may exist. The PROCESS HANDBOOK supports this selection process [47, 329]. In WORKWARE, these dimensions of reuse are integrated through the decision construct, simplifying the overall framework while increasing the flexibility (allowing other relations that XOR and AND). WORKWARE also adds a process perspective. The process of appropriating a template model to your local process is thus an unfolding sequence of:

- Reuse decisions, the selection of template models at ever finer levels of detail,
- Adaptation by adding and removing model elements. These elements may also have template specialisation hierarchies.

By representing this as a process, we capture the current state of reuse decision making. All decisions need not be made at the start of the project, instead the selection among candidate templates can be postponed right up to the time when these items are to be activated. This framework is thus far more flexible than full-template-copy approaches. There is nothing in this design that is special for process models. Reuse of product structures, organisational models, and conceptualisations can also be handled in this way. However, the metaprocesses are especially useful for structures that evolve while they are operational, e.g. product models during design, organisation structure models during reorganisation, and dynamic domain ontologies [260].

Explicitly modelled reuse decisions also create opportunities for automated selection of templates, e.g. based on properties of the current model. For instance, a selection among more and less detailed procedures can be made automatically depending of the skill levels of the persons allocated to perform the work. In product models, dependencies between components and the environment can also be utilised for automatic component selection. For instance, if a product is to be placed outdoors, non-corroding materials are chosen. In complex, multi-dimensional selection tasks such as design, simple hierarchies of alternatives will not suffice. The interplay of many aspects demands multiple, overlapping and evolving structures.

6.2.2 Resource Allocation Decisions

The decision tree involved in template selection has a parallel in resource allocation. Tools, personnel, information and material objects must be selected among a number of candidate role fillers. In some cases, the allocation decisions can be made during initial process modelling. Other decisions, e.g. personnel allocation, are more contextual, and must often be resolved at runtime. Figure 62 shows fragments of an organisation model. The company SINTEF has a number of institutes (3 shown). Each institute has departments (3 departments of the Telecom and Informatics institute is shown), and the departments may have groups (Systems technology, CSCW and HCI shown). Employees have a role in a group. Figure 62 shows the default resource allocation of SINTEF personnel to the EXTERNAL project. The CSCW group is the one participating in the project, and four of the five people of that group play roles in the project. In the model, activated candidate relations are visualised with thicker lines than those not selected. For inter-organisational projects, this yields a number of benefits:

- A common organisational model is reused. Placed in a decision-making context, the organisational units become "typecasted" as OR decisions.
- Partners see where the people they collaborate with are located, and get an overview of the rest of the organisation's capabilities.
- When a workitem is assigned to SINTEF in the context of this model, it is by default interpreted as "one of the people at SINTEF allocated to EXTERNAL". This is achieved through reuse of the model in Figure 62 for all items inside the project.
- The final assignment to a workitem is represented as selecting one of the activated candidates. Such a selection deactivates the candidates not chosen, leaving only the actually allocated personnel.
- In the case of exceptions, e.g. if the project needs expertise not possessed by the allocated people, or if they do not have time to do all the required work, the model provides an overview of alternative resources. For instance, someone from the HCI group could be involved in requirement specification, and someone from the Systems Technology group might be called upon to solve technical problems. Such exceptions are handled by activating more candidate relationships.
- Policies of personnel allocation often rely on organisational hierarchies, in that each manager is responsible for the allocation of employees reporting to her. The organisational model reflects this by making local colleagues alternatives at the nearest level. Exceptions in resource allocation are thus treated as backtracking along the decision steps in the organisation hierarchy. The structure is also useful for defining resource allocation policies, who has the rights to assign which persons to what kind of work.

Other types of resources may also be allocated along decision trees. For software tools, personal preferences are important. If the objective of a workitem is to produce a web page, which HTML editor tool to apply should be selected based on the preferences of the person responsible. This implies that a predefined decision tree regarding software tools preferences of the individual is reused along personnel resource allocation relationships. A user interface for defining such preferences can be dynamically generated based on available alternatives. Similar solutions may also be applied for managing and selecting information resources and document templates.

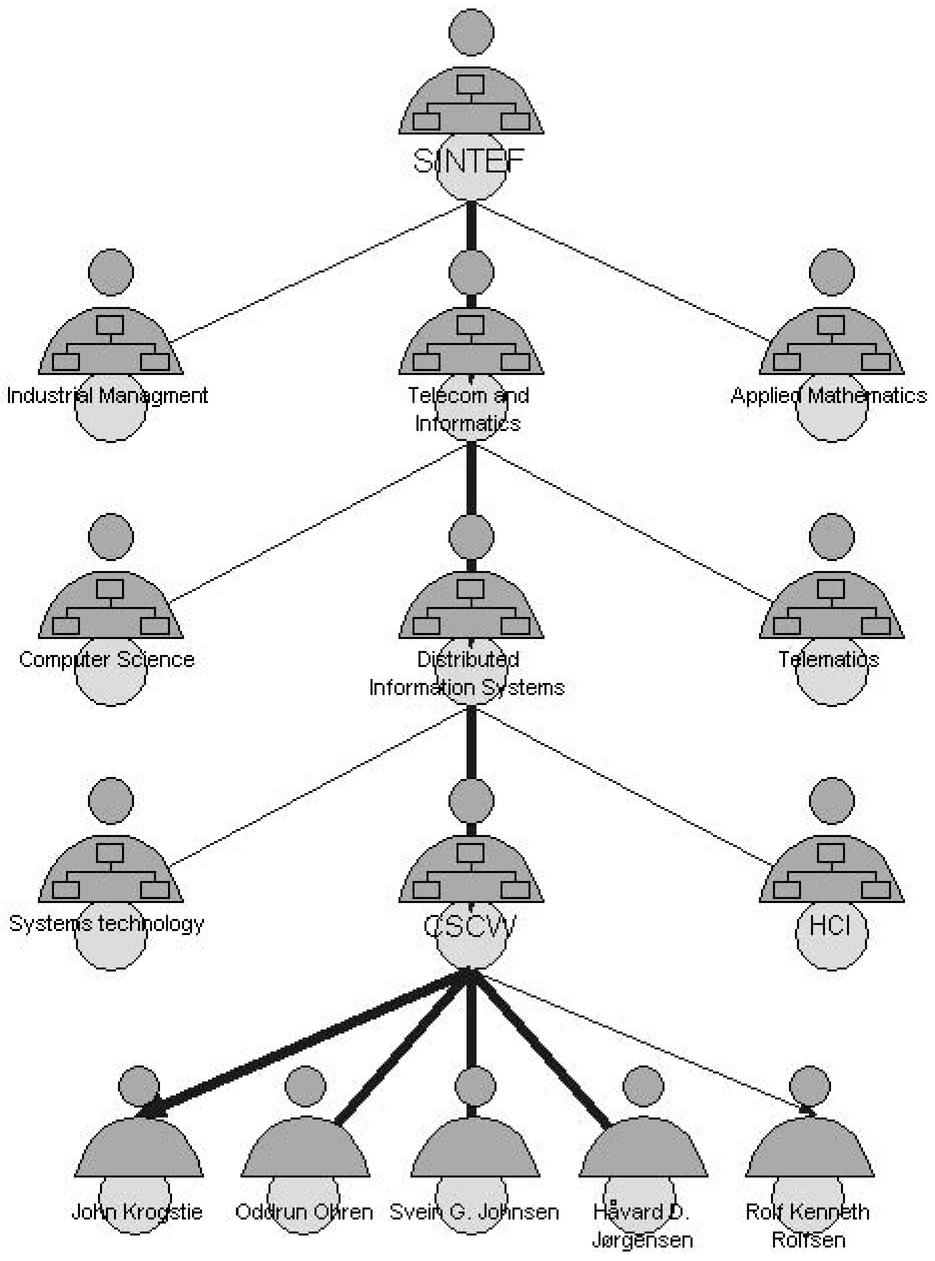


Figure 62. Organisational model with allocation decisions for a particular project.

Although these composite decisions are rooted in multi-layered models of candidates, it is easy to limit the user interface for decision making to the current level. For instance, when a new task is allocated to "SINTEF" in Figure 62, the user interface could present a list of the four candidate individuals, for the users to select among. In addition, a button for widening the scope of candidates (moving up one layer at a time, backtracking the latest decision) could be included. When the allocation decision is made, the result is just an attribute, e.g. "Responsible = Håvard D. Jørgensen", of the workitem in question. Rich models thus need not result in overly complex user interfaces.

6.2.3 Product Design Decisions

So far we have seen that process, organisation, tool and information structures can be activated using decision constructs. Products seem no different. During design, production and maintenance, selections must be made between alternative components, and various exceptions can arise that requires the current selection to be re-assessed. Explicit representation of decisions thus has a role in product modelling as well. This is exemplified by the existence of tools for capturing design rationale and decision processes in information systems engineering (cf. section 3.1.3).

6.2.4 Decisions in Classification According to Concept Ontologies

Brasethvik and Gulla [70] utilise concept models for document classification and retrieval. Document classification involves tagging documents with metadata (e.g. keywords) so that they can be more easily retrieved. This classification is semi-automatic, in that a natural language processing component selects candidate keywords from the text. Users can then add new keywords, remove those that do not fit or add more specialised terms. Classification can thus be seen as a decision process that utilises the concept model. Specialisation relations are treated as OR-decisions, AND represents synonyms, while antonyms and homonyms can be modelled as XOR decisions.

6.2.5 User Interface Adaptation and Tailoring Decisions

Alternative configurations of user interfaces constitute another domain where decision modelling can play a part. A user interface consists of a number of components. Some are containers that include other components. Again we may support user interface customisation as a combination of AND (container content) and XOR (alternative components) decisions in a tree. WORKWARE presents model content through customisable user interfaces. A list interface may include one container for each object, each containing one component for each attribute. The user interface customisation process thus becomes a selection of which components to use for which attribute. An example of how this can be implemented is presented in Chapter 7.

6.2.6 Decision Making Processes

Underlying all of the examples discussed in this section is a common model of how the decision making process can be captured. This model is rather simple and does not capture the human and social complexities of actually arriving at a decision. The model results in structures similar to decision trees, which have a long history in management and computer science [482]. Briefly summarised, the decision making process as captured by these elements is a modular one, which for each decision object, include the following states[7]:

1. Decision not made, none of the output candidate relations are active.
2. Candidates identified, some of the output candidates are active, but the decision is not final (the decision object is not activated itself).

[7] These states are compatible with the state transition diagram for workflow decisions (Figure 34 on page 100).

3. Decision is made (decision object and the selected output candidate(s) are activated). This step causes the state of the not-selected candidates to be set to *not activated*. When a decision is made, all previous decisions on the path leading down to it may also be marked as activated.
4. *Backtracking* may occur. This corresponds to increasing the scope of alternatives by undoing a previous decision. Backtracking is needed for exception handling.

For resource allocation and template selection decisions, an active candidate relationship, which comes out of an activated decision connector, is treated as an is-filled-by relationship for that resource role. The discussion in section 4.6 on rules, decomposition, timers, and metamodelling for decision automation can be generalised from workflow decisions to other kinds of decisions. It is also possible to go one step further and model *decision making processes* with the WORKWARE PML. This can be captured as decomposing a decision into a process of workitems, or by simply including workitems as part of a decision graph or tree. Typical examples include resource allocation workitems and product design workitems.

6.3 Semantic Holism

The gap between the real and the virtual, between phenomena and their representations, is a fundamental problem in computer science [190, 480]. Situated, contingent, uncertain, open and complex social reality is hard to capture in formal, computerised models [189, 455]. The discussion of research methodologies in Appendix A points to a similar gap between reflective practice and the linear, formal models of science. Because we cannot observe everything that goes on in a system, causal relations appears uncertain, circular and non-linear [447, 524]. In Chapter 2 we saw that these problems by no means are limited to scientific models. Outsiders' articulation of work processes often does not capture the complexities of actual work performance [468].

This section shows that some of these limitations stem from the conventional notion of atomic semantics, that each model element should be defined by itself. Semantic holism is thus proposed for making computerised representations more user-oriented. With semantic holism, the meaning of each model element may depend on all the other elements in the model [55, 242]. This concept underlies most of the modelling techniques proposed in this thesis. Since semantic holism is quite new in computer science [263], we start with an exploration of its roots.

6.3.1 Semantic Holism in Philosophy, Linguistics and Informatics

In analytic philosophy the challenges of understanding the real world and how we talk about it, has spurred the development of semantic holism [55, 242, 243]. The *Routledge Encyclopedia of Philosophy* defines semantic or mental holism as

> *"The doctrine that the identity of a belief content (or the meaning of a sentence that expresses it) is determined by its place in the web of beliefs or sentences comprising a whole theory or group of theories"* [55].

Molecularism determines meaning and content in terms of small parts of this web, while *atomism* defines elements independently of the rest of the web. Atomism thus claims that sentences have meaning independently of their relations to other sentences or beliefs. It has been argued that holism has a number of weaknesses [55]:

- It makes generalisation difficult, because sentences cannot be fully understood outside of their context.
- It makes it difficult for multiple theories to share any sentences or beliefs, because the meaning of those beliefs are contextualised in each theory,
- It makes logical reasoning, agreement, and translation difficult,
- It makes the semantics unstable, in that any change in one's attitude towards a sentence will change the meaning of the terms contained in it. This is called the *instability thesis* [243].

These arguments have been met by replacing the dichotomy between agreement and disagreement with a gradient of similarity of meaning. In a holistic perspective the degree of similarity of meaning can have multiple dimensions. *Two-factor theory* is another interesting approach [55]. Here the meaning of a sentence consists of an internal holistic factor and an external referential factor that relates terms to real world entities. Jackman [243] further proposes a moderate version of holism to avoid instability. In his framework the mapping from beliefs to meaning of a sentence is many-to-one. This implies that although a change in beliefs *potentially* can alter the meaning of a sentence, every belief change need not *actually* affect the meaning.

Semantic holism is evident in a number of areas. Social construction of meaning in communities of practice [43, 527] emphasise the local meaning of terms inside a community and the role of boundary objects in cross-community relations. Boundary objects have a common identity across communities but also a specific local meaning inside each community [462]. Boundary objects thus reflect the two-factor theory. Psychology teaches that conceptualisations arise not out of individual sensory perceptions but emerges from the whole history of the individual's experiences [55]. In biology, ecosystems and genomes are described in holistic terms [194].

Most IS modelling languages are formal or semi-formal, and atomic semantics dominate. Typically one tries to identify model elements with a clearly defined interface that captures all the element's relations to its environment. This *constructive* approach results in a division of the problem into smaller sub-problems so that when all the parts are solved, so is the whole. Holism challenges the feasibility of a constructive approach when faced with messy and complex realities. Interactive models, immersed in practice, are especially vulnerable to this critique. There are also some aspects of holism in conventional IS languages. Inheritance causes the meaning of one element (the subclass) to depend on other elements (superclasses). The reuse framework presented in Chapter 5 goes one step further in the direction of semantic holism by enabling inheritance along all links.

6.3.2 Semantic Holism in Work Processes

Let us illustrate semantic holism with a simple work process example. Figure 63 shows a model of a work process with only one step, *Write project application*, carried out by a research scientist. In order to improve the quality of proposals and limit the legal liabilities of the company, the organisation in question later decided that a manager must review all applications. The new procedure is depicted in Figure 64. Interpreting these models with atomic semantics, one would claim that the task of writing the application has not changed, as both its definition and its interface remains the same. If, on the other hand, we were to interpret the models from a holistic perspective, the semantics of the writing workitem is altered, because it is put in a new context.

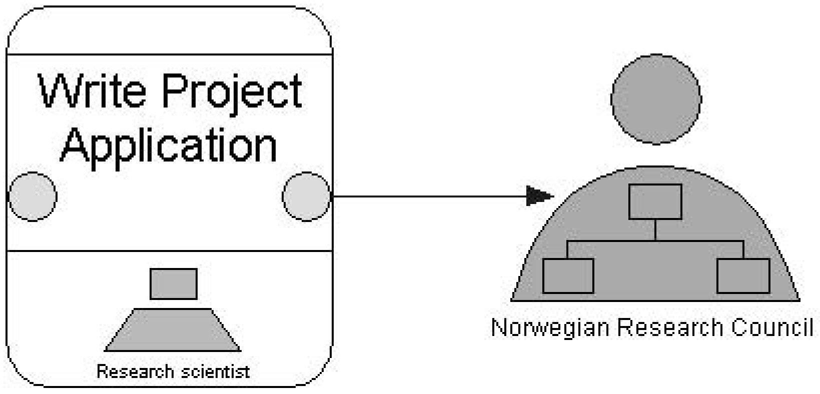


Figure 63. A simple work process: Writing a project application.

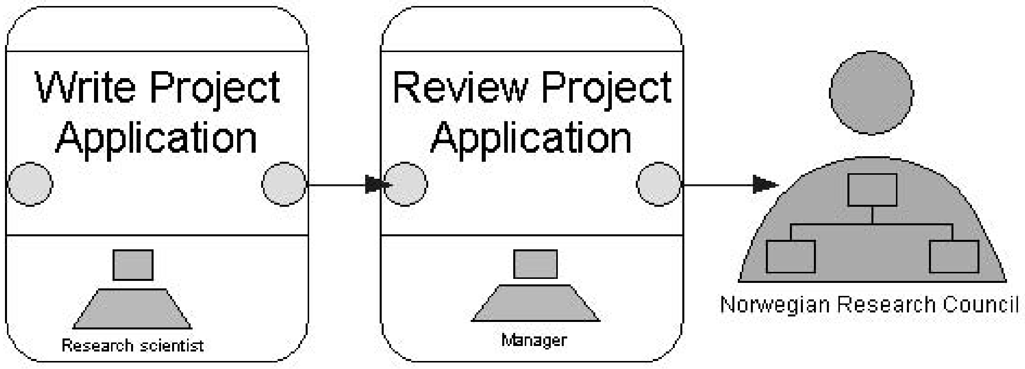

Write Project Application
Research scientist
Review Project Application
Manager
Norwegian Research Council

Figure 64. A revised work process, involving an additional review.

The holistic interpretation seems to better match the view of human actors. When the application you write does not go directly to the research council, but first to an internal review, this influences the way the application is written. For instance you have to complete the application some days before the deadline of the council, in order to allow time for the review. You may also choose to discuss the application more with your manager, so that you are sure she understands and will accept it, or to make sure to include any details that the manager expects. These tacit dependencies extend beyond direct neighbour elements.

Later, the model in Figure 64 was revised yet again, decomposing the review into two sub-items, *Assess budget*, performed by the finance director, and *Assess risks*, performed by the head of the department (Figure 65). Although these sub-items are internal to *Review application*, and thus due to encapsulation should not affect *Write project application* at all, semantic holism allows them to. Again, this reflects more accurately the interpretation of the procedure by the people involved. They know that nowadays they should discuss the application with the finance director as well before submitting it. A conventional automated interpretation of this model, e.g. by a workflow system, would follow atomic semantics. Focused at relieving users the burden of coordination, such systems do not discriminate two identical workitems put in different contexts. From an automation perspective, it is beneficial to divide the process into steps with well-defined interfaces capturing all interdependencies. From a human perspective, information about the context of work is crucial for sense-making, motivation, and coordination. Since the model does not capture everything that may be relevant, semantic holism gives a richer, more accurate interpretation of what the model represents.

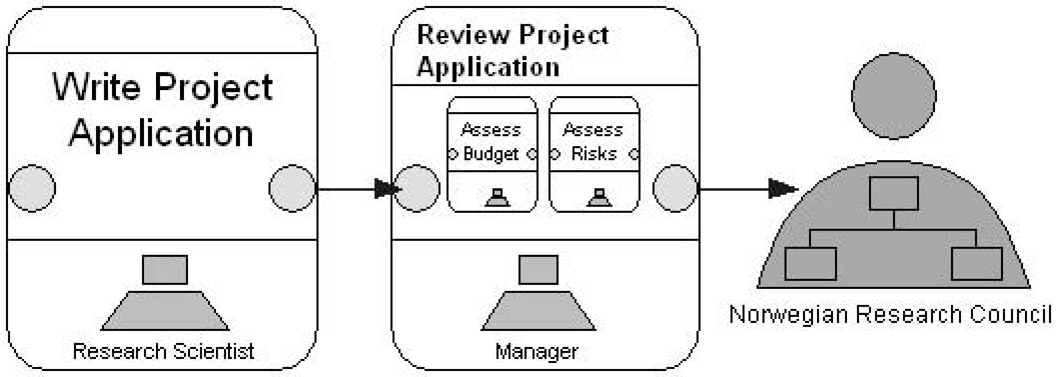

Write Project Application
Research Scientist
Review Project Application
Assess Budget
Assess Risks
Manager
Norwegian Research Council

Figure 65. Revised model for project proposal review.

6.4 Semantic Holism Simplifies Model Articulation

Semantic holism also more closely reflect the meaning of natural language than formal, atomic approaches. This is reflected in Functional Grammar (FG) [83], which captures the meaning of natural language statements in a structured way. In FG a sentence is made up of a number of elements, each contributing with different aspects to the meaning of the sentence [83]:

- *Predicates* denote entities, properties and relationships. Predicates have *semantic functions* in sentences, e.g. agent, positioner, force, processed, state, possessor, goal, or argument.
- *Predications* instantiate the predicates into a spatiotemporal location (location, time, duration, frequency). They may also describe participants, means and manner, consequence, purpose, motivation, cause, explanation, polarity (negative or positive), tense (past, present, future), epistemic (certain, possible, impossible) and deontic modality (obligatory, permissible, forbidden).
- *Propositional content* adds modalities (subjective, evidential, or objective) and attitudes (personal evaluations of the statement).
- *Clauses*, denoting the *illocutionary speech acts* implied by the statement (declarative, interrogative, imperative, or exclamative), as well as *pragmatic* functions relating to the process of communication.

The meaning of a sentence is thus expressed by a combination of all its elements, and the meaning of each element depends on the others in *sentence holism* [397]. FG also deals with inter-sentence relationships in discourse analysis and rhetorical structures [83]. The referents of predicates often rely on other sentences to be resolved. In IS research, both Klein [273] and Kangassalo [263] note that translation from one language to another requires a holistic understanding of sentences as well as whole texts and their cultural context. Ricœur [410] describes how literature, by creating an imaginary world, expresses meaning through connotation rather than denotation and reference. Communication must establish relations between two worlds, the sender's and the receiver's. In small groups, interactive two-way communication establishes such relations. One-way broadcasting relies more on established conventions and shared reality in a common culture.

Reality as we observe it, natural language and communication have clear holistic features. Semantic holism thus seems a promising strategy for human-oriented modelling languages. By combining the meaning of different terms and clauses, users can articulate more nuance knowledge than with atomic formalisms. Through adding and removing parts of a sentence, flexibility, ambiguity and uncertainty can be captured. Natural language, made up of just a few tens of letters and other symbols, can express far more knowledge than UML's more than 200 constructs [281]. Although some degree of structure and formalism is needed for automation support, we can still learn a lot from natural language when it comes to increasing the efficiency of modelling languages. In this section we therefore investigate techniques that make languages more holistic.

6.4.1 Limited Classification

To simplify the modelling language by limiting the classification of primitives, was a design principle that WORKWARE inherited from APM [90]. In these languages we only have one construct that represents a unit of work (workitem or action). In most workflow management systems, there exist a wide variety of concepts for units of work. For instance, the WfMC standard includes these variants [531, 532] (Figure 10 on page 50):

- *Process,* a non-atomic class concept,
- *Activity,* an atomic class concept, which is further specialised into
  - *Manual activities*, and
  - *Automated activities,*
- *Process instance,*
- *Activity instance,*
  - *Work item,* instances of manual activities,
  - *Invoked application,* instances of automated activities.

Such a large vocabulary is useful in order to create a precise standard for process definition interchange. As a modelling language for business users, it is unnecessarily complex. In WORKWARE all of these concepts are represented as workitems, and the meaning expressed by the specific constructs of the WfMC, is *derived from the context* in which each item is placed:

- Processes are workitems that are decomposed, while activities are not,
- Workitems that are part of a template are "class" models, while items placed in a local model are instances,
- Automated and manual items are separated by whether the resource signature includes an invoked software tool or not.
- Further specialisation of workitems can be expressed by adding new properties, e.g. controlling the enactment rules.

In addition to greatly simplifying the modelling language, WORKWARE's scheme is also more flexible. In a typical lifecycle of a workitem, decompositions may be added or removed. When we separate processes from activities, these typical evolution patterns require the objects to change class. Such migration is difficult in many class-oriented systems [24], since classification is an inherent part of the definition of the object [393]. Conversely, the lack of any inherent separation between templates and instances enables WORKWARE to use the same mechanism for reuse-by-copy as for reuse-by-instantiation.

Conceptual simplicity is crucial for the usability of reuse. Even trained software engineers sometimes have problems sorting out multiple meta-levels, and for end users it is even more difficult.

Even though the basic language does not require the detailed vocabulary of the WfMC to be explicit, it may be convenient for articulation if templates are offered for different kinds of workitems. Such templates would be defined as a *constellation* of a few objects, e.g. an *automated activity* template includes a workitem with an *invoked software tool resource role*. If the user later changes his mind and decides that he must do the work manually instead, he just has to remove the tool resource, and need not redefine the whole workitem.

Decisions are another area where the principle of limited classification was applied in the design of WORKWARE. APM ports, conditions, timers, combiners, and splitters were grouped together into one construct, based on the realisation that they all represent decisions regarding the flow of work. Here context (input, output or neither), properties (e.g. deadline for timers) and relations (number of flows in and out) allow us to separate among different subtypes. Any concrete WORKWARE model can thus be translated into larger languages such as the WfMC standard [532].

6.4.2 Instances are Referential Terms

The two-factor theory of semantic holism points to an interesting feature of instance modelling: Instances are referential terms; their aim is to directly identify real world entities or phenomena. In the case of workitems, flows, resources, and decisions, we are not talking about physical entities, but rather of concrete work that the users involved (hopefully) assign meaning to. At the very least, a meaningless workitem has a context where the process of negotiating its meaning can be performed. For instance, the model identifies other stakeholders that can be contacted in order to discuss what the work involves. Instance modelling thus facilitates human and social interpretation of the meaning of the model. When modelled instances have a direct representation in the user interface of software tools, the reference from model element to the online workspace should also be straightforward to grasp for most users.

6.4.3 Properties Modularise Aspects of Meaning

Similar to the way predications, propositional content, and clauses are added to predicates in Functional Grammar, dynamic addition of *properties* to modelled objects allows users to articulate facts and meanings in a flexible manner, ranging from very detailed and precise (long) sentences to ambiguous and contextual (short) statements. Variations in the degree of model specificity seems simpler to articulate through adding (or removing) properties than by reclassifying instances according to a new understanding of what they represent. Properties thus simplify model evolution compared to class-oriented approaches.

One example of how this can be utilised is the modelling of resources. Resource roles are abstract as long as they are not filled, i.e. as long as the role-filler property is not set. When the property is set, the abstract resource role becomes a concrete assignment. Properties are also utilised to separate among different kinds of objects. For instance, in section 5.5.5 we saw how different enactment policies could be applied to workitems depending on their properties. Another example would be to let properties denoting uncertainty, vagueness and incompleteness (as in SeeMe, cf. section 3.2.4) affect the way workitems and decisions are managed.

6.4.4 Derived Properties Enable Contextual Semantics

Derived properties constitute a simple mechanism for contextual semantics. A derived property is automatically computed from values in the model object's environment. The various types of workitems discussed in section 6.4.1 are separated by properties like *composite*/*atomic*, *generic*/*specific*, and *automatic*/*manual*, which can be derived from the context of each individual item. All roles that are (possibly recursively) filled by a resource, represent this link as a derived property. This means that if the role *Responsible* on item *Design* is filled by the role *Chief designer* on the workitem *Software development project*, and the role *Chief designer* is filled by *Tom*, both the roles have a derived property called *filled-by* pointing to the person *Tom*. These derived properties are utilised, e.g. in the user interface, where users during enactment normally just cares about who fills the role and not about how this was achieved in the model. By mixing derived and *articulated properties*, we also allow users to say directly who fills a role (by assigning a value to the filled-by property) as an alternative to modelling a resource allocation process. Elaborate schemes thus co-exist with simple, direct mechanisms. Through derived properties any model element can, in principle, influence any other element. This constitutes true semantic holism. The reuse framework presented in Chapter 5 can implement derived properties.

6.4.5 Constellations

Sentences can be articulated by grouping objects together in a *constellation* (or assemblage [466]). A workitem is a constellation of resources, input and output flows and possibly sub-items modelled in a workflow. The workitem itself corresponds to the predicate part of a sentence, while the resources are the subjects and objects involved. Flows denote temporal order, and possibly goals or reasons. The more objects one adds to such a constellation, the more precise its meaning becomes. Generalising this principle, one could also view an object as a constellation of properties (treating properties as first class primitives). Property modelling thus becomes a particular kind of constellation modelling.

The individual objects in a constellation can also derive meaning from the whole. Figure 66 shows person objects placed in a number of different constellations. The upper line shows generic roles (with no *filled-by* property). Placed on its own a person object denotes a generic individual. Placed inside a workitem the same object denotes a *role* in that item. Inside an organisation, the generic person object represents a *position*, and multiple persons put together denote a *group*. You may also place a person role on a *flow*, denoting that the source should allocate resources for the target. For all of these different constellations, generic roles can be filled by concrete individuals, as shown in the lower line in Figure 66. Modelling by constellation thus facilitates a systematic approach to personnel management across domains.

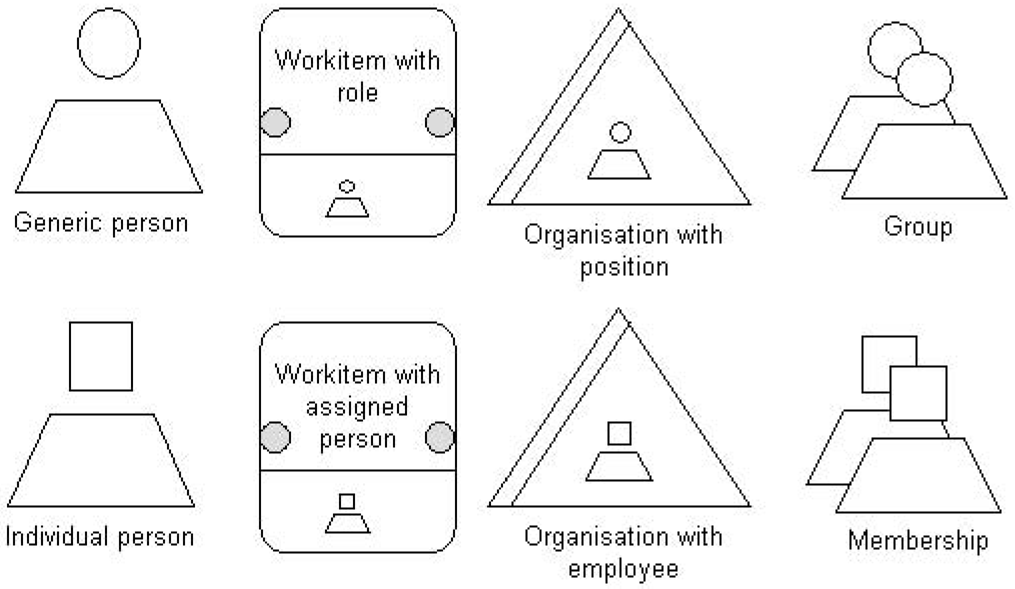

Workitem with role
Generic person
Organisation with position
Group
Workitem with assigned person
Individual person
Organisation with employee
Membership

Figure 66. Different uses of Person objects.

Many modelling languages employ separate concepts for every one of these uses of person objects [347]. While this makes sense when different features are needed for each of the contexts, it also makes the modelling language larger and less flexible. The resource modelling framework of APM (Figure 21 on page 84), contains a multitude of variants of actor (organisational, external agent, or software agent), tool (manual or software), and object (material, information, active information object, or pluggable action) resources. Most of these resources can be either role or concrete, invoked (mandatory) or available, and composite or atomic depending on their properties. The total number of resource variants include 9 basic and 4 generic types, times 8 (2 by 2 by 2) property variations, in total 104. Clearly all of them cannot be represented as primitive classes. The number of primitive properties required to articulate the same amount of information, is by contrast 8. The number of primitives is thus decreased from equalling the *product* of the number of property values (for classification) to becoming the *sum* of different values (properties and constellations).

Each different person object in Figure 66 may require specific features that are not needed for the other cases. For instance, while positions may have salary codes, roles have hourly rates. But there will also be a number of common aspects, for instance those concerned with allocation (as shown by the two lines in the figure) or related to the person who fills the role. Semantic holism dictates that the object definition (in this case the person object's set of properties) may depend on the context in which the object is placed. The reuse framework presented in Chapter 5 allows users to define policies that inherit properties from the context (e.g. workitem, organisation or group) to the person objects, implementing this form of semantic holism. The dynamics of model evolution further dictates that the object definition should depend on the states of the elements in the model, For instance, while generic person objects may have skill requirements, individuals have skills. Figure 67 shows an example where the semantics of the person role depends on indirectly associated elements (work processes and organisations respectively). Both parallels and differences between these two model fragments were discussed above.

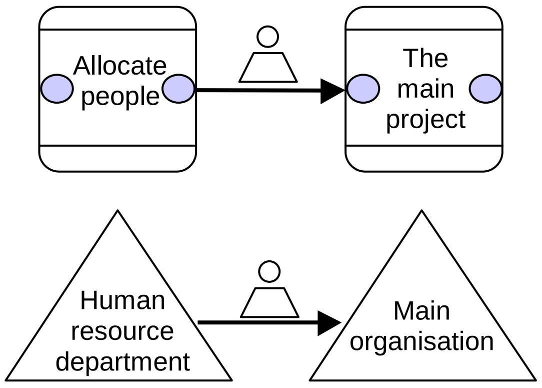


Figure 67. Personnel allocation to organisations and work processes.

Styhre [466] discuss concepts in organisations and management science. He notices that concepts interrelate with practice in unstable and ambiguous ways, and emphasises concepts that are *multiplicities*, conjunctive synthesis of singularities (AND-combinations of instances). Such concepts get their *meaning from relationships* between the elements. Styhre shows that concepts such as human resources and total quality management should be understood in terms of a wide range of singularities (such as flexibility, skill, process) from different disciplines [466]. Some process and enterprise modelling languages similarly have emphasised relationships as the central modelling construct [241, 309].

6.4.6 Constructive Composition vs. Holistic Interdependencies

In software development, constructive decomposition of systems into subsystems is a fundamental technique. In a constructive structure, overall system properties can be computed when subsystem properties are known. Only constructive structures can be validated [482]. Development proceeds through recursive divide and conquer of each subsystem, until a set of implementable basic components are defined. Constructive approaches also emphasise separation of concerns in decomposition structures [392], maximising the internal cohesion and minimising external couplings of each part. Available systems and components are reused through bottom-up synthesis.

Open, interactive systems are different. As Wegner [523] points out, correctness of an open system cannot be validated in general, only with respect to a well-defined set of inputs. Interactive models reflect wicked, incompletely understood problems, which has not yet been tamed, framed and solved. They support the problem solving *process*, not just the documentation of its outcome. Constructive composition yields control, but complex and changing social environments makes it impossible to predefine all connections between subsystems. Several researchers find that maximum adaptability is achieved on the "edge of chaos" [194]. Black-box, closed, constructive structures should thus be replaced by open, reflective implementations [139, 268] and semantic holism. On the other hand, social conventions, laws, contracts, standards, and technological decisions also close subsystems with respect to features that clients rely on, allowing simpler systems [299]. The degree of subsystem openness thus involves a trade-off between simplicity and flexibility. Kizcales et al. [269] list different types of open interfaces:

- The environment may declaratively describe its intended use of the component so that it can select a matching implementation strategy. With semantic holism, these domain properties can be derived from the surrounding model. We call this *contextualisation*.
- The environment may explicitly decide among a set of implementation strategies offered by the component, such as WORKWARE's interactor policies. This is called *parameterisation* [41] or *customisation* [488].
- *Extension* [488]: The environment may itself define parts of the subsystem implementation strategy, e.g. user-defined enactment rules.
- *Adaptation* and *specialisation* [139]: Strategies may also be incrementally redefined. In WORKWARE, property and constellation modelling support incremental articulation, while cancellation inheritance allows existing strategies to be replaced.

The core challenge for open implementations is to manage complexity. In our case, complexity is managed interactively, allowing different roles (users, process experts, software engineers) to adapt the system through the interfaces listed above. *Scope control* should be natural and sufficiently fine-grained [269], which instance modelling and explicit inheritance policies facilitate. Meta interfaces should be *orthogonal* to primary interfaces, and default policies should make tailoring *optional* [269]. The aspect-oriented WORKWARE architecture consists of orthogonal interactors, and default activation policies have been implemented. However, in an interactive model the separation between domain and implementation strategies is fuzzy. Semantic holism thus violates the orthogonality requirement. Process models are especially ill suited for constructive approaches. Based on experience that most changes affect several of the process steps, Parnas [392] concludes, "*it is almost always incorrect to begin the decomposition of a system into modules on the basis of a flowchart*". Our architecture with multiple model interactors thus decomposes code by concerns rather than by processes.

6.4.7 Summary

This section has shown a number of aspects of semantic holism in interactive modelling languages, using practical examples from WORKWARE. Semantic holism gives rise to these principles for modelling language design:

1. Limit the use of classification to encode information.
2. Let instances establish direct links between model elements and what they represent.
3. Allow properties to be dynamically assigned to objects.
4. Allow constellations of objects to be manipulated as an entity.
5. Allow inheritance and derivation of properties along any link.
6. Never explicitly state facts that can be deduced from other parts of the model.

Combined, these techniques generates modelling languages that are *simpler* (with fewer primitives), more *flexible* (accommodating typical model evolution scenarios), and *richer* (allowing a larger, extensible set of nuances of meaning). We have also seen that semantic holism better matches natural language and the way human actors perceive the world. Thus semantic holism should be suitable for model articulation by end users. Table 5 summarises the many gaps between the way users perceive the world and how this is normally represented in information systems.

Reality as observed by users	**Objectives of IS models**
Partially understood	Complete
Holistic causality	Linear causality
Non-deterministic	Deterministic
Contextual, holistic meaning	Constructive composition, well-defined interfaces
Open system	Closed system
Emergent, decentralised order	Designed, central control
Satisficing, adequate solutions	Optimisation, "best practice"

Table 5. Gaps between real world semantic holism and IS modelling reductionism.

6.5 Semantic Holism in the Activation of Models

Holistic activation semantics view a model as a system of autonomous components [524]. Each component can be formalised, but their interaction, controlled by users, cannot. Hence, the system exhibits *emergent behaviour*, which is non-deterministic, decentralised, and incompletely predefined [194]. However, order and stable patterns also emerge from the interaction of autonomous components in open, *self-organising* systems [357]. Even though each individual component has a simple, formal definition, interaction and openness yields a rich system behaviour. Semantic holism, employing contextual interpretation rules, combining the states of all elements to decide the situated meaning of a model, amplifies and balances emergent behaviour.

6.5.1 Interactive and Holistic Workflow Architecture

The enactment semantics of WORKWARE interconnects the state transition behaviour of multiple model elements. Each element offers interaction capabilities so users can trigger state transitions that complement or override rule-based interpretations. Users may also add or remove elements as the enactment progresses. Hence, a WORKWARE model is an open system of interacting, semi-autonomous elements. Semantic holism is evident in a number of features:

- State transition rules refer to other objects, often through multi-step navigation, thus the interpretation of the current object depends on related objects. By triggering actions and state changes in other elements, the indirect consequences of a transition can potentially reach through the whole process structure.
- Interaction sequence diagrams (cf. section 4.5.4) reflect that the overall semantics of a model emerges from the interaction of simple, independent rules for each element.

Figure 27 (on page 92) presented a logical architecture of an interactive workflow system, consisting of a shared workflow model, a number of model interactors, and an integrated user interface. In Figure 68 this architecture is revisited. It shows multiple users interacting with multiple model elements. Similarly, the system contains several cooperating model interactors, each offering a partial interpretation of the model. The holistic interaction metaphor thus applies to all three levels of Figure 68.

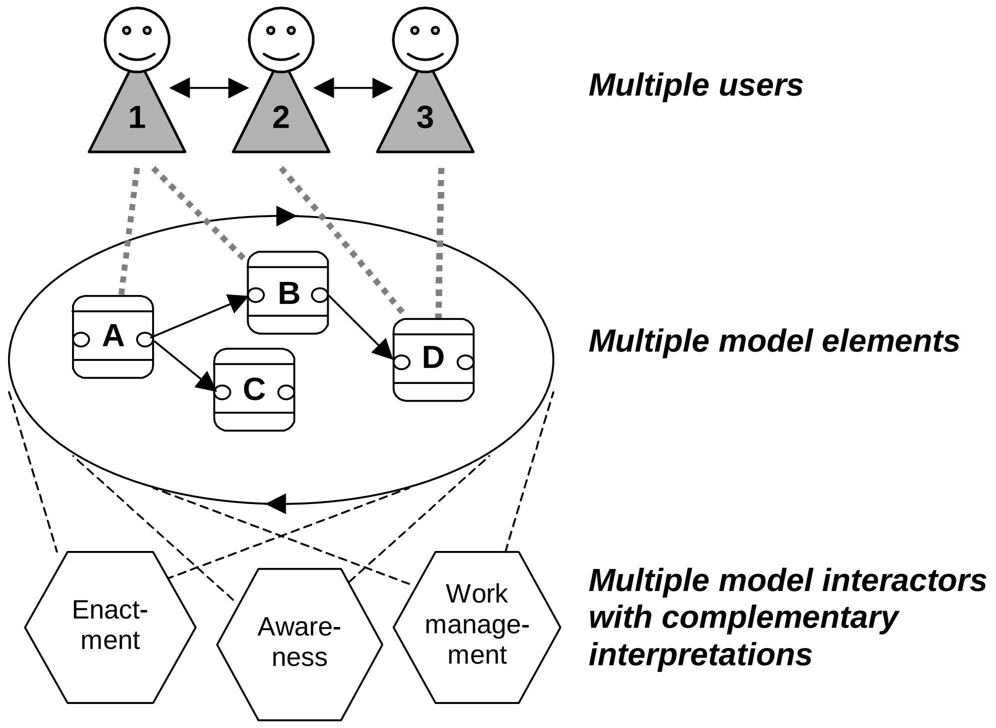


Figure 68. Holistic activation semantics.

6.5.2 Holistic Interaction among Users

An information system can be viewed as a medium for human interaction, communication and knowledge dissemination. Among systems that support cooperation, those that offer basic support for human interaction without sophisticated automation, e.g. email, chat, and shared workspaces, are most widely used [42]. When a user is looking for a solution to a problem, it is not important whether the answer comes from a computerised component, a database (or another asynchronous communication medium), or directly from another user. The Turing test, whether a user is able to distinguish answers from a computer from answers from a human being, is of little practical relevance [523]. Instead we need a system that supports the social processes of knowledge creation, dissemination, codification and learning. Joint sense-making, articulation of reference frames and contexts, are important aspects of these processes. This requires interpretative flexibility of the knowledge representations. Hence formal languages without contextual meaning are not appropriate.

6.5.3 Holistic Interaction among Model Elements

An interactive model consists of elements that are directly and indirectly connected. When users interact with model elements in manners not narrowly rule-restricted, each element becomes partially autonomous with respect to the rest of the model. Since users can add and remove objects and relationships, the population of model elements is evolving. By utilising as much as possible of the knowledge that is articulated in the model, holistic activation mechanisms can better provide contextualised support. The many different interaction scenarios that can be generated from the state transition models in section 4.5, illustrate the potential for a system of simple components to exhibit and mediate complex behaviour.

Comparing WORKWARE's enactment semantics with that of Petri nets [553], we see clear differences. In Petri nets only the transitions that currently have tokens on all of their input places are candidates for activation [535]. The rest of the model is not utilised in any way to contextualise the interpretation. The example presented in section 6.3.2 illustrates the need for such contextualisation, as does the case studies surveyed in Chapter 2. Petri net semantics works well for formally correct, complete, static models, but cannot handle incompleteness or openness. Petri net model evolution requires complex mechanisms that are hard to understand for end users [7, 152, 153, 548, 553]. Table 6 and Table 7 illustrate these differences. It is based on a simple model with two workitems A and B planned to occur in sequence (A before B). The rows show the states of A, while columns show the states of B. Neighbouring cells thus show possible state transitions. In the Petri net example there is only one way through this model. All empty cells are illegal. (Basic Petri nets do not separate Ready from Ongoing).

	State of B →			
State of A↓	*Waiting*	*Ready*	*Ongoing*	*Finished*
Waiting	Input place of A receives token			
Ready	↓			
Ongoing	All input places of A have tokens			
Finished	Transition A fires when user is finished	→	Input place of B receives token	Transition B fires

Table 6. Petri net enactment of model with two workitems A and B in sequence.

	State of B →			
State of A↓	*Waiting*	*Ready*	*Ongoing*	*Finished*
Waiting	Input flow activated	B declared ready before A, exception	User starts B before A, early start exception	B finishes before A, violation of modelled sequence. Strong warnings should be issued
Ready	All input flows activated			
Ongoing	User declares A started	Flow is manually activated, so B becomes ready before A finishes	Opportunistically increased concurrency	
Finished	User declares A finished	Flow from A to B activated	User declares B started	User declares B finished

Table 7. WORKWARE enactment of model with two workitems A and B in sequence.

For WORKWARE all of the cells are filled. Although we planned to perform A before B, exceptions may reschedule the work. Such exceptions are tolerated; thus they are captured inside the system and can be the subject of later reasoning and learning from experience. The system also supports the users in managing these exceptions, e.g. the awareness engine helps the performers of A and B to maintain an overview of what goes on in the other task. But even in the normal case of operation, WORKWARE allows more situations to occur than Petri nets. Users may for instance trigger an outgoing flow (from A to B) before the item (A) is finished, if all that is needed for the start of B is already produced in A. Languages that attempt to handle the scenarios above based on closed system atomic semantics, need a lot of different flow relationships [183]. They must separate *B-must-finish-after-A* from *B-cannot-start-before-A-finishes* etc. Interaction and semantic holism, on the other hand, allows us to handle all the cases with a rather simple language, and it lets users delay the decision of detailed flow semantics until the situation arises. If users want to predefine detailed flow semantics, they may add specialised enactment or access control policies to the flows.

The notion of model activation as holistic interaction among model elements applies to other kinds of interactive models as well. Between product components or concepts in a corporate ontology the relationships are many, and probably incompletely articulated. The Gossip awareness engine for software product models [163] is one example where this is implemented

6.5.4 Holistic Interaction among Model Interactors

The third level of activation which can be described as a system of interacting components, consists of the interactors that mediate user interaction with model elements and activates the rules of their concern. In Chapter 4 we saw how the interactors of WORKWARE provide complementary coordination support controlled by process models. Different interactors support different scenarios. We also saw how changes to the model made by one component affected the others. This flexibility is achieved because no interactor assumes it has full control of the model.

This design is extended in the EXTERNAL infrastructure (Chapter 7), where WORKWARE is integrated with a model editor, a real-time cooperation tool and a process simulator. In such an environment all of the components become more useful than they are in isolation. The model editor not only visualises static models of typical work processes, it can also show overviews of the current state of actual processes. Similarly, the simulation tool now can use data from ongoing projects to adjust its parameters and provide more accurate forecasts. The integration of more model interactors, offering a richer set of functionality customised by the current state of work, is an important direction for further research. This integration also increases the utility of the interactive models as a medium for knowledge transfer among users. By integrating different tools around a common model, we also create an arena where different work practices can meet (management, knowledge workers, accounting, quality control etc.). At the same time this integration is problematic because different communities of practice assign different meaning to the same terms (e.g. boundary objects as discussed in Chapter 2). Precisely because different communities (and their tools) assign different, yet partially overlapping meaning to the elements of a shared model, we need open and contextual semantic holism. Semantic holism allows shared model fragments to change meaning depending on the context (e.g. local practice-specific views) in which they are placed, and can thus accommodate different uses by different communities. By allowing these complementary interpretations to co-exist and interact, semantic holism allows models to fill the role of boundary objects, facilitating communication and learning within and across communities of practice. Instance modelling creates a global identity for each element, maintained across all communities. Property modelling and incremental classification allow different communities to articulate their own local aspects of these objects in an integrated manner [385]. Additional objects with a local meaning may influence the interpretation of shared elements through derived properties and reuse policies. This implies that e.g. the human resources (HR) department in Figure 67 can have a far more elaborate model of the personnel recruitment process than the main organisation, but nevertheless the effects of the HR activities on the shared model are available to all.

6.6 Semantic Holism in Model Reuse

The model reuse component from Chapter 5 can be regarded as just another model interactor. By handling inheritance along any modelled relationship, it extends the degree of semantic holism offered by conventional systems. By assuming that models are evolving and incomplete, the framework also follows open systems perspectives. Rather than supporting a single, centrally controlled inheritance hierarchy, WORKWARE recognises that decentralised emergence better matches the needs of end users, as illustrated by the examples in Chapter 5. The lifecycle of model evolution (section 2.5.7), points to a number of roles for semantic holism:

- *Adaptation and appropriation* of a template fragment into a local model can be partially automated if the meaning of the template is influenced by the new context (the local model) in which it is placed. Through derived properties and reuse policies a template fragment can automatically adapt to its new situation, increasing the potential for reuse without the need for tedious adaptation by the end users. For instance if we add a *Review* template workitem to a local process, it could automatically inherit all the document resources of the local process, which are the basis for the review.
- *Generalisation* similarly benefits from semantic holism. Derived properties are automatically reset when a model fragment is moved from its local context and into the template repository. By inheriting features from the template context, e.g. states set to initial values, resources roles not filled etc., generalisation gets a head start. Different contexts for organisational, group, project and personal templates allow different generalisation rules to be applied.
- *Classification* of templates involves a mix of predefined hierarchies and need-driven structures that emerge from the properties of local models. Classification structures help users see similarities and differences among candidates. The mere experience of seeing such structures may influence the users' understanding of each template. The context that classification provides can also enhance this process, e.g. highlighting differences among a current set of candidate templates, pointing to common features etc. When a user makes a partial selection, e.g. "I like these features in the template, but these other features do not fit our project", such functionality would be especially useful. The context utilised by semantic holism in this case is the selection process represented in Figure 58 and Figure 60 (pages 137-138).
- *Analysis and evaluation* of the reusability of a model could similarly be supported by specialised tools, reusing quantitative properties associated with model elements, e.g. time and resource consumption, costs, perceived quality ratings etc.

6.7 Summary of Contributions

Chapters 4, 5, and 6 have described the contributions of this thesis, encompassing a general approach for flexible IS design (interactive models), and specific ideas for process support environments and other interactive, model-driven solutions. The table below maps the contributions to the core challenges uncovered in Chapters 2 and 3. It shows that the new design ideas presented here address the challenges. The validation in Chapters 7-9 indicates that these design ideas are indeed feasible, useful and innovative.

Research challenge	Contributions that meet the challenge
Incomplete models	Interactive activation, explicit decisions
Language efficiency (simplicity)	Semantic holism, instance modelling
Flexible articulation	Semantic holism, property modelling
User-oriented language	Metamodelling, instance modelling, holism
User involvement	Interactive activation, location of decisions
Contextual model interpretation	Holistic activation semantics
Rich functionality	Multi-interactor architecture, semantic holism
Customisation	Policy models, contextualisation
Reuse as an interactive process	Reuse processes and decisions interactively enacted
Need-driven reuse	Emergent classification (by intension and extension) in the metametamodel
Customisable reuse	Reuse framework with policies and aspects, generalised inheritance
Metaprocess support for reuse	Reuse processes and decisions integrated in practice, removing the "meta" character

Table 8. Research challenges and the contributions that meet them.

Chapter 7
Implementation

The WORKWARE prototype has been developed and experimented with in a number of research projects since 1997. It implements most designs from Chapters 4 and 5, serving as proof of concepts. The original objective of WORKWARE was to support planning, performing, coordinating, and managing ad-hoc work, as well as learning from practice. As the work progressed, it was recognised that ad-hoc processes also have substantial routine parts, and that structure emerges as the work progresses. Discussions with users and the previous research surveyed in Chapter 2 showed that seemingly routine work also has strong ad-hoc and knowledge-intensive characteristics. Consequently it became important to integrate the support for routine and emergent workflows within one framework.

7.1 WORKWARE Principles

In addition to the research-oriented requirements presented in Chapter 2, WORKWARE had to satisfy a number of detailed user needs, technical as well as functional requirements. These additional requirements are presented first in this section. We then outline core design principles, user interface and system architecture, to provide an overview of the system's functionality and services.

7.1.1 Requirements

The next chapter describes usage experience with WORKWARE and the integrated infrastructure of the EXTERNAL project. The most important requirements put forward by these cases are:

- The system should help each user organise his work. When organisations install groupware tools, early adopters often discover that real benefits only can be reaped when most of the people they work with also use the tool [202]. By providing services that users need individually, we can overcome this *critical mass* problem.
- Different aspects of work should be integrated, not separated. Planning, coordination, management and reporting are integrated in, not external to, knowledge work.
- The system should support visual modelling, but not require it. Users should also be able to articulate their work inside the work environment, e.g. in textual forms.
- Simplicity and ease of use is a primary concern, especially for novice users. Migration from novice to expert should be facilitated.
- Customisation is needed to allow individual user preferences, variation in organisational routines etc.
- Evolution should be handled and facilitated, both of work practices and technological support. The system should easily adapt to new platforms. Different client access devices, software, and network bandwidths must be supported.
- It should be easy to start using the tools. Client installation should be straightforward,

and the system should be useful even without a detailed process model.
- Information should be shared among users, in a groupware manner, not distributed and hidden from others, as in personal mailboxes.
- Standards should be used to integrate external services and tools. Cooperation with partners that are not using the system should be supported.
- Information should be stored and transferred in a secure manner, preventing unwanted disclosure to external as well as internal actors.
- Performance should be satisfactory, matching user expectations to similar technologies.

In addition, a number of requirements would be important for a stable software product, e.g. scalability, locking and versioning. WORKWARE was however designed as an experimental prototype, so these requirements were not prioritised.

7.1.2 Design Principles

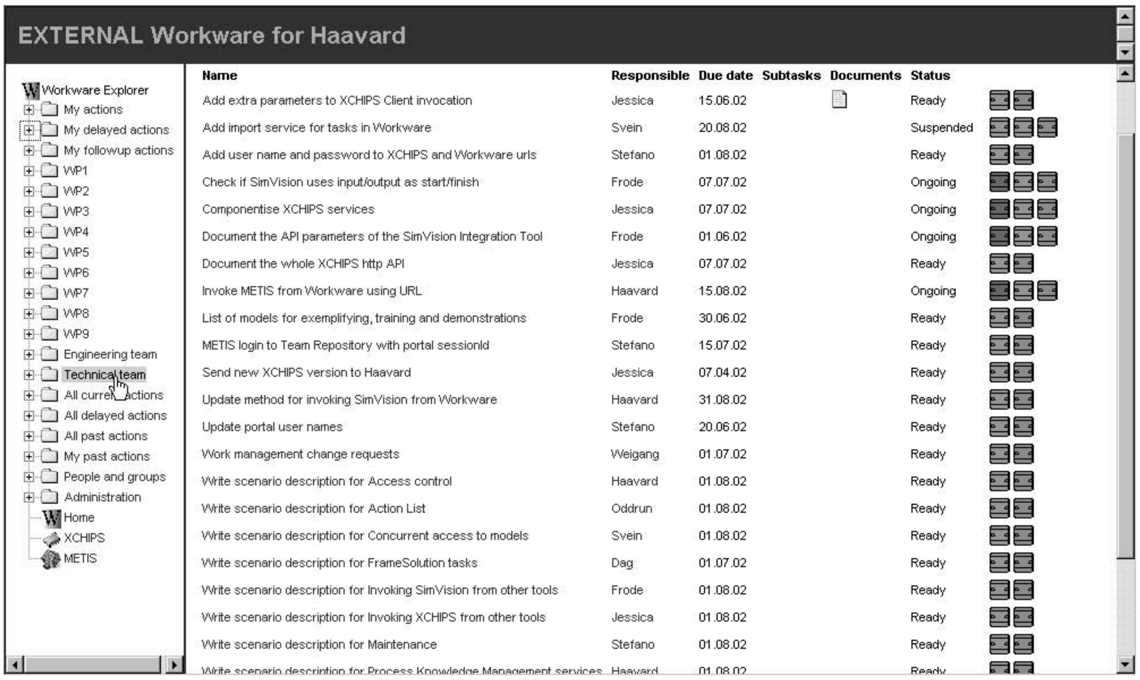
Based on the requirements, these principles were early on articulated for the software architecture and development process:
- The system should be web-based. The client should run in a web browser and not require installation of extra software. The system should not use proprietary features.
- The user interface should be dynamically generated in order to facilitate personalisation and model-driven customisation.
- The system should be component-based. A strong separation of concerns should exist between the components.
- Available standard and open source components should be utilised. Built-in browser capabilities should be utilised for desktop application integration.
- The implementation process should be incremental to provide early benefits for the users. Basic services like information sharing and task management, should be the first objectives, while more powerful features could follow later.

Based on these principles, Java Servlets [474] were selected as the main implementation technology. At the time, it offered the most comprehensive framework for dynamic web applications. The user interface consists of dynamically generated HTML forms and documents, available across HTTP from the web server that runs the servlets.

7.1.3 User Interface

In Chapter 4 the user interface components of WORKWARE were briefly introduced. The two main forms for interacting with models of work, are the *worklists* and the *worktops*. Worklists presents overview of workitems according to selection criteria specified by the user organisation, including:
- Who fills roles (as responsible, participants, or customer) on the item? These criteria are used to separate personal and group worklists.
- Which project, work package or parent workitem does the item belong to (derived from the work breakdown structure)?
- What is the state of the item (e.g. separating new items from ongoing ones)?
- Is the item delayed?
- Time. What is the (planned and actual) start and finish date for the item? Is it scheduled for this week of far into the future?
- Any additional property defined by the user organisations can also be used.

In addition to criteria for searching the database of workitems, a user may explicitly add or remove items from a list, overriding the search selection. Each worklist displays a customisable selection of properties for each item. Figure 69 shows an example.


EXTERNAL Workware for Haavard

Workware Explorer
- My actions
- My delayed actions
- My followup actions
- WP1
- WP2
- WP3
- WP4
- WP5
- WP6
- WP7
- WP8
- WP9
- Engineering team
- Technical team
- All current actions
- All delayed actions
- All past actions
- My past actions
- People and groups
- Administration
- Home
- XCHIPS
- METIS

Name	Responsible	Due date	Subtasks	Documents	Status
Add extra parameters to XCHIPS Client invocation	Jessica	15.06.02		📄	Ready
Add import service for tasks in Workware	Svein	20.08.02			Suspended
Add user name and password to XCHIPS and Workware urls	Stefano	01.08.02			Ready
Check if SimVision uses input/output as start/finish	Frode	07.07.02			Ongoing
Componentise XCHIPS services	Jessica	07.07.02			Ongoing
Document the API parameters of the SimVision Integration Tool	Frode	01.06.02			Ongoing
Document the whole XCHIPS http API	Jessica	07.07.02			Ready
Invoke METIS from Workware using URL	Haavard	15.08.02			Ongoing
List of models for exemplifying, training and demonstrations	Frode	30.06.02			Ready
METIS login to Team Repository with portal sessionId	Stefano	15.07.02			Ready
Send new XCHIPS version to Haavard	Jessica	07.04.02			Ready
Update method for invoking SimVision from Workware	Haavard	31.08.02			Ready
Update portal user names	Stefano	20.06.02			Ready
Work management change requests	Weigang	01.07.02			Ready
Write scenario description for Access control	Haavard	01.08.02			Ready
Write scenario description for Action List	Oddrun	01.08.02			Ready
Write scenario description for Concurrent access to models	Svein	01.08.02			Ready
Write scenario description for FrameSolution tasks	Dag	01.07.02			Ready
Write scenario description for Invoking SimVision from other tools	Frode	01.08.02			Ready
Write scenario description for Invoking XCHIPS from other tools	Jessica	01.08.02			Ready
Write scenario description for Maintenance	Stefano	01.08.02			Ready
Write scenario description for Process Knowledge Management services	Haavard	01.08.02			Ready

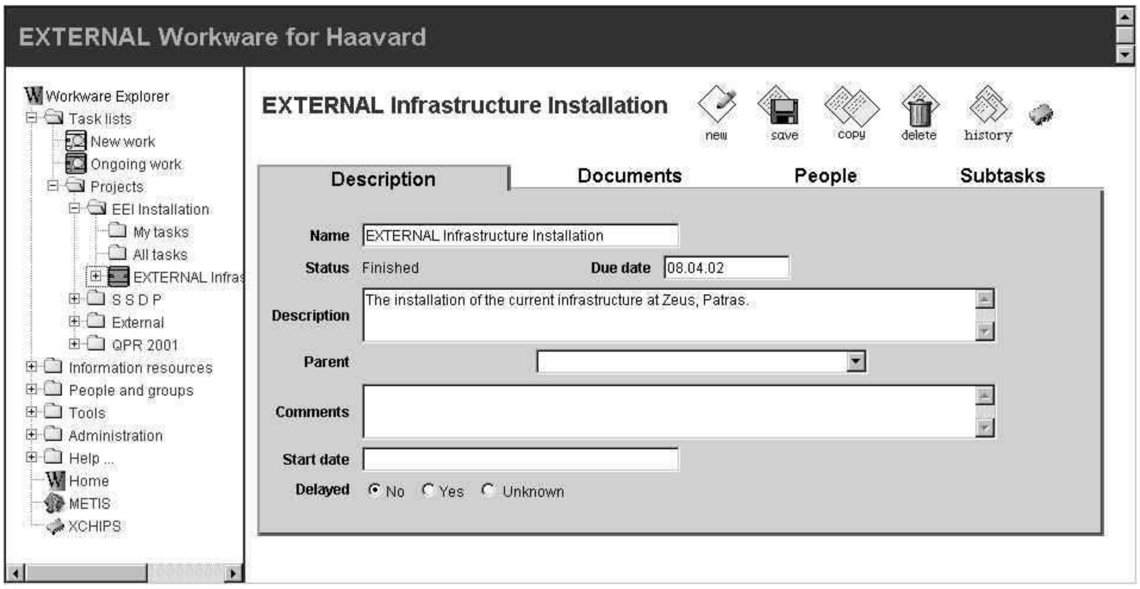
Figure 69. WORKWARE worklist.

By clicking on the name of a workitem in the list, the user opens the *worktop* of that item. Figure 70 shows an example. The worktop includes a description of the item, communication links to the people that fill roles, links to document resources, and a worklist containing sub-items. The users also have access to three types of *services*:

- *Planning services* (articulation), including process modelling,
- *Performance services*, providing access to tools and information,
- *Coordination services*, reporting on work progression, e.g. changing the state of a workitem or selecting output flows.

Services correspond to commands that the user can invoke on the workitem. They are work steps so fine-grained that we do not represent them as workitems by themselves. The classification into planning, performing and coordination services refers to the use of a service in a particular context; the same service may be used for planning, performing or coordination in different work items. A planning service like "Process modelling" can for instance be used to perform the workitem "Adjust project plan".

In addition to these components for interacting with workitems, the system also includes general forms and services for manipulation of other types of objects, as well as specific interfaces for some types, e.g. decision-making forms for user involvement in the activation of decision connectors. All of these forms, and all the components and services they contain, can be customised according to organisational policies and user preferences. The WORKWARE Explorer fills a separate frame to the left of the main forms. It is a menu-structure that organises services hierarchically according to the preferences and access rights of the individual users. Menu configurations can also be shared (reused) among users. More details about how services are selected and customised to meet local needs, are provided in section 7.3.1.



EXTERNAL Infrastructure Installation

Name	EXTERNAL Infrastructure Installation
Status	Finished
Due date	08.04.02
Description	The installation of the current infrastructure at Zeus, Patras.
Parent	
Comments	
Start date	
Delayed	[✓ No] Yes Unknown

Figure 70. WORKWARE worktop.

7.1.4 Architecture

Below, the architecture of WORKWARE is described at three levels. The *logical* architecture (section 7.3) shows the main components, the *implementation* architecture (Appendix B.1.1) defines packages of code, while the runtime *deployment* architecture (Appendix B.1.2) contains servers, clients, and related software systems. At the logical level, the prototype currently integrates five model interactors, a work management tool, a workflow enactment engine, an awareness engine, a document manager and an access controller. WORKWARE is further integrated with tools for visual modelling, simulation, and real-time collaboration in the EXTERNAL infrastructure (section 7.4). The implementation is divided into 4 layers of Java code, for user interface, interaction control, data management, and persistent data storage. At runtime the WORKWARE Java servlets are connected to a web server. The servlets build HTML forms and documents dynamically. These interfaces are returned to the web client, via the web server, using http. The set of user interface forms and components has evolved throughout the lifetime of the system. Appendix B.4.2 shows five generations of WORKWARE user interfaces, starting with rudimentary HTML forms and ending with a multi-frame, iconic interface controlled by style-sheets. We have also applied different server operating systems and web servers. The architecture has handled this technological evolution well.

7.2 Implementation of the Language Metametamodel

At the data management layer, each WORKWARE server has one *DataCatalogue*, which gives access to all modelled objects. This layer implements the metametamodel defined in Chapter 5:

- *Classes*. For each class, clients can get a list of all the members. Classes are identified by a unique name.

- *Templates*. (One for each class). New objects are created as clones of templates.
- *Objects*. Object instances are identified by a unique id.

In addition to the DataCatalogue, the main implementation class of WORKWARE's data management layer is the *DataObject*. It contains operations for access to the name, type, implementation (a Java class), help text, and attributes of a model object. Relationships between objects are encoded as attributes that has a DataObject class as its type. *List-DataObjects* allow multi-valued attributes. Each list has a default element type.

7.2.1 Reflection

WORKWARE's data management scheme applies reflection to simplify user interaction and increase flexibility. Classes are treated as objects. The class *Type* has all classes as its members, and thus serves as a metaclass. Type can fill all the roles of ordinary classes. This implies e.g. that attributes can have classes as their values. This feature is useful for customisation, e.g. when one wants to define policies for all objects of a certain class. The object that defines one such policy typically has an attribute denoting its scope, and its value may refer to a class. *Attribute* is another metaclass. It is e.g. useful for defining user interface policies for how individual attributes should be displayed. Table 9 shows a policy that defines how the Description attribute of Workitems is to be displayed for a particular user.

Attribute name	**Attribute type**	**Example value**
Profile for type	Type	Workitem
Profile for attribute	Attribute	Description
Profile for user	Person	Håvard
User interface component	Component	Text input field (3 lines)

Table 9. User interface policy that utilises WORKWARE's reflection mechanisms.

These mechanisms are highly useful for configuring the general functionality of the system. For instance, the general form for editing objects, enquires what attributes the current object contains, and what their types and values are. This information is then used to select customised user interface components for each attribute.

7.2.2 Metamodelling

WORKWARE's reflection capabilities are also used for metamodelling. Since classes themselves are treated as objects, defining a new class is just a specialised way of defining a new object. WORKWARE currently support these kinds of classes:

- *Basic classes* are defined with an existing object as their template. The template is edited and manipulated just like other objects.
- *Extensional classes* are defined by normal objects. The extension template is a List-DataObject with Type as its element type. Users add subclasses just like they add elements to other lists.
- *Intensional specialisations* are also defined by ordinary objects. The template has attributes that contain the selection criteria to be applied.
- *Enumerations* are defined by a ListDataObject template.

Class metamodelling is most useful at the group and organisation levels. For most end users, metamodelling at the instance level is more appropriate. Since the set of attributes (names and value types) are defined locally for each object instance, users may add (or remove) properties as they see fit (subject to access control policies). The case studies presented in Chapter 8 include several examples where this has been utilised.

7.2.3 Assessment

As shown in Appendix B.2.3, this implementation utilises most design patterns for flexible data management [173]. The detailed design thus seems to be well aligned with state of the art. The metametamodel has been in operation since 1997, and has proved capable of handling a number of scenarios beyond those it was originally defined for. Examples are provided below.

7.3 Implementation of Interactive Activation

The model interactors are the components that activate process models in WORKWARE's architecture. Typically such a component includes the following parts (Figure 71):

- *User interface* components that allow users to participate in model interpretation,
- *Model access* through DataObjects that the interactor uses and updates.
- *Event subscription* interface where the interactor registers with the objects that it works on. Whenever an object is created, deleted or changed, interactors are notified and react according to their interpretation of the current state of affairs.
- *Policy models* (ordinary DataObjects) that control the behaviour of the interactor.

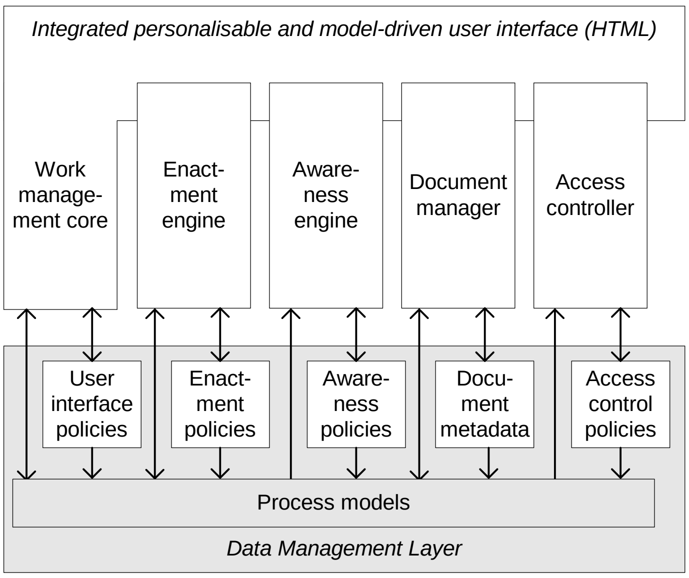


Figure 71. WORKWARE interactor architecture.

7.3.1 Work Management

This core component of WORKWARE implements generic mechanisms for sharing model data among distributed users, providing simple, customisable forms for viewing, editing, listing and searching the modelled data. The core thus supports generic articulation and manual activation of models of any kind. Specialised interfaces (worklists, worktops, decisions making forms) have been implemented for process models. The work management tool controls most of the user interface to the system, integrating components and services from other interactors where appropriate.

Multi-Level Configuration of Services

WORKWARE models units of functionality as *service* objects. In addition to the system's own services, external web services and desktop tools can be integrated. The set of services included in each user interface is determined dynamically by combining *Service-Configurations* for several aspects that describe the current situation:

- The classes of the current object,
- The operation mode, e.g. edit or view, list or single object,
- The current user, enabling personalisation, and
- The current object instance.

ServiceConfigurations may exist for each of these aspects and for some combinations. The configurations are applied in sequence, starting with the most general (superclass) and ending with the most specialised (instance and user). Each configuration can both include and exclude services, implementing *cancellation* inheritance [483]. Such multi-level customisation has proven very flexible, enabling both general evolution and local preferences. Figure 72 illustrates the application of this scheme. Here three different projects have customised the explorer menu to include the services that they need.

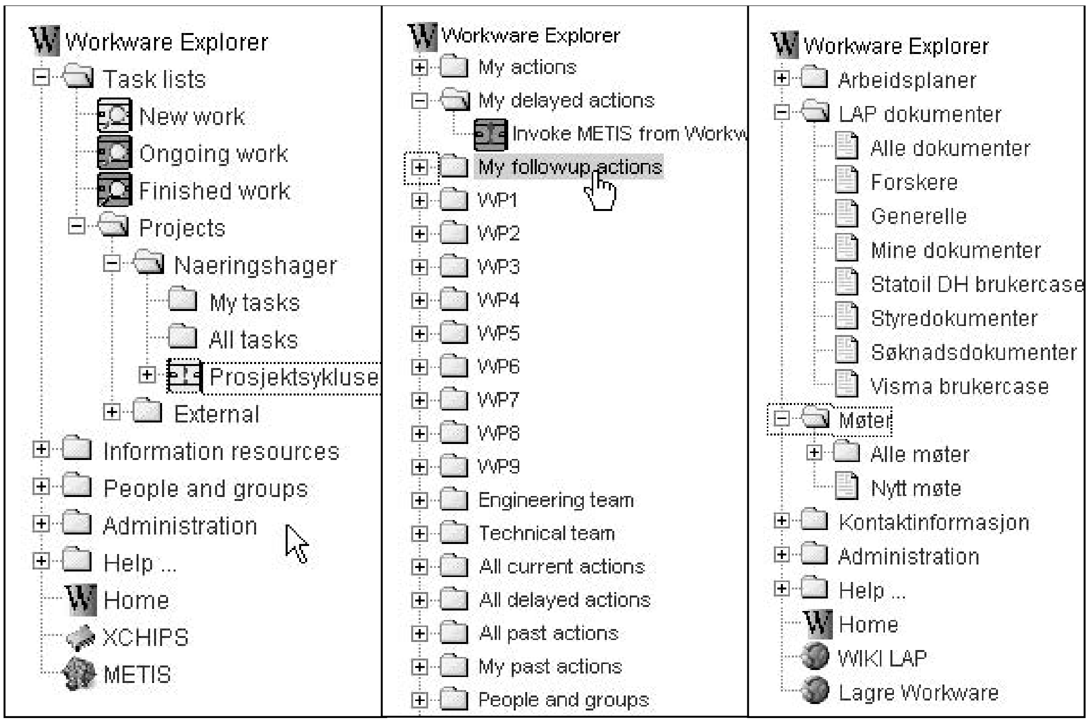


Figure 72. WORKWARE Explorer menus for different usage contexts.

The configuration to the left is the default, while the second one is tailored to support management and coordination of action items in a project. Actions are ordinary workitems, but presented here under a different name to fit the local vocabulary of the users. The organisation of the project into work packages (WP) and cross-functional teams, is mirrored in the menu structure. It was also chosen to highlight delayed actions, and to use the customer role to denote follow-up responsibility. Lack of follow-up was a reported problem in current practice. In the third example, the language has been changed to Norwegian (by renaming the service objects). Meetings have been added as a special subclass of workitem (defined by intension), and different categories of documents are distinguished by a local, user-defined property. These examples thus show both the metamodelling, modelling and service configuration aspects of customisation. More cases will be presented in Chapter 8.

7.3.2 Interactive Enactment

The interactive enactment engine subscribes to change events on all workitems, decision connectors and flows. When the state attributes of these objects change, the engine updates related objects according to the rules defined in Chapter 4. The engine is mainly an automation tool, but it also includes some specialised user interface components. The *CoordinationServiceList* lets users update the state property of workitems. It includes services for triggering the state changes that are allowed in the current state, determined by the outgoing transitions in the state transition diagrams. Figure 69 on page 165 shows CoordinationServiceLists for each workitem under the heading *Status*. This enactment component is included in the work management interface because it is the default view style for attributes of type *Workitem status* according to modelled user interface policies.

7.3.3 Awareness

Rolf Kenneth Rolfsen originally developed the awareness interface for the SEASPRITE infrastructure, which allows organisations to share lifecycle information about ships, structured by product models [374]. For WORKWARE we mapped process structures to his generic awareness model [416]. We also added the use of filtering lenses [50, 518] to customise holistic awareness mediation along the relations of the process model (cf. section 4.7). The awareness engine is built with standardised interfaces for event capture and propagation, and is thus not limited to work process models. All objects that have event logging enabled, are members of the extension class "Objects with event log". Policies define that awareness related services are included when members of this class are displayed.

7.3.4 Document Management

The principle behind this interactor is to utilise process models to classify documents, in order to simplify retrieval, management and contextual interpretation of the information. The component was originally designed as an example of ontology-driven information workspaces [365]. It is thus extensible to other kinds of interactive models as well. A simplified version was implemented in WORKWARE, utilising open source software. The example menu to the right in Figure 72 shows that the metametamodel of WORKWARE can support and utilise other forms of metadata on documents as well. In this case a *document category* attribute was used for separating documents in the menu. However, WORKWARE is not a full-fledged document management system. Lack of versioning and a cumbersome user interface, are among the limitations of the current implementation.

7.3.5 Access Control

The overall concept and services of the model-driven access control component were presented in section 4.8. The access controller remedies some of the information management problems mentioned above by integrating an open source content management system called JAKARTA SLIDE [17]. This system supports distributed authoring and versioning according to the WebDAV standard [235]. In addition to storage, retrieval and access control, locking and versioning are also supported by SLIDE. These services were integrated into WORKWARE through a component that listens for changes on all the model objects, and makes sure that these changes are reflected in the access control structures. We also provide an interface for defining access control policies with user-oriented concepts (the modelled projects, workitems, users, groups, and documents). SLIDE supports inheritance of access rights in the directory hierarchy. More details about how this feature is utilised for reuse are presented below.

7.3.6 Summary and Assessment

Unlike object-oriented frameworks, WORKWARE's architecture separates behaviour from data. Even features commonly thought of as inherent to the data structures, e.g. inheritance, is managed by model interactors and not by the data management layer. The motivation for this design is to allow user interaction and customisation. The design groups related behaviour rules in a manner similar to aspect oriented programming (cf. section 5.5.8) [157, 351]. For interactive process models aspects include enactment, awareness, access control, resource management, data management, visualisation, and user interaction. The mismatch between aspect and object oriented perspectives is evident in the UML models of activation semantics in Chapter 4. Here the behaviour rules managed by one interactor are spread across a large number of modelled objects, even for very simple scenarios. Aspect-oriented modelling frameworks [193, 385] that support behavioural as well as structural modelling could solve some of these problems.

Like *join points* or *weavers* [193, 473] integrate code for different aspects into a running system, the model objects and their event notification interface integrates different interactors in WORKWARE. Communication through data structures creates a loosely coupled system where each of the interactors can work in isolation. The interactors do not directly invoke operations from one another [48]; they are orthogonal [44]. This pluggability has been utilised to support different system configurations, e.g. with or without the EXTERNAL infrastructure tools. Many user benefits however arise from the integration of previously independent services around a common model, so technical flexibility should not be over-emphasised. Integration of many different aspects into one model can make it complex and unmanageable. WORKWARE includes a number of remedies for this problem.

- Each interactor need only care about the objects, classes and attributes that it uses. Other features can be ignored and hidden from function-specific user interfaces.
- Each interactor typically defines some specific object classes (policies) for customisation. Other interactors need not bother with those. Of the 100 classes installed with a standard WORKWARE server, 10 deal exclusively with service customisation, 31 with user interface configuration, 10 controls awareness mediation, and 16 support reflection and data management. The remaining interactors typically apply a few policy classes each, while less than 20 core classes are shared among the interactors.

Since WORKWARE is an interactive system, our aspects also include user interface components. User interface integration thus becomes a key problem. The generic user interface layer of WORKWARE does not solve the problems of integrating different technologies. As the next section shows, such integration was difficult across the tools of the EXTERNAL infrastructure. This thesis does not address problems of technical interoperability, but shows the potential for model driven user interfaces to integrate functionality from different interactors in a contextual and customisable manner, provided that technical interoperability is in place.

7.4 The EXTERNAL Infrastructure

The EXTERNAL project [161] aims to facilitate inter-organisational cooperation in knowledge intensive industries. It is the hypotheses of the project that interactive process models form a suitable framework for tools and methodologies for dynamically networked organisations. The EXTERNAL infrastructure (Figure 73) integrates a number of tools for articulating and activating process models:

- METIS [309], an open, general purpose enterprise modelling and visualisation tool,
- XCHIPS [233], a hypermedia tool with process and real-time collaboration support,
- SIMVISION [295], which simulates processes and resource allocation,
- WORKWARE,
- UEPS [491] (User Environment Portal Server), a general environment for Internet and Intranet portals,
- FRAMESOLUTIONS [93], a framework for automated workflow applications.

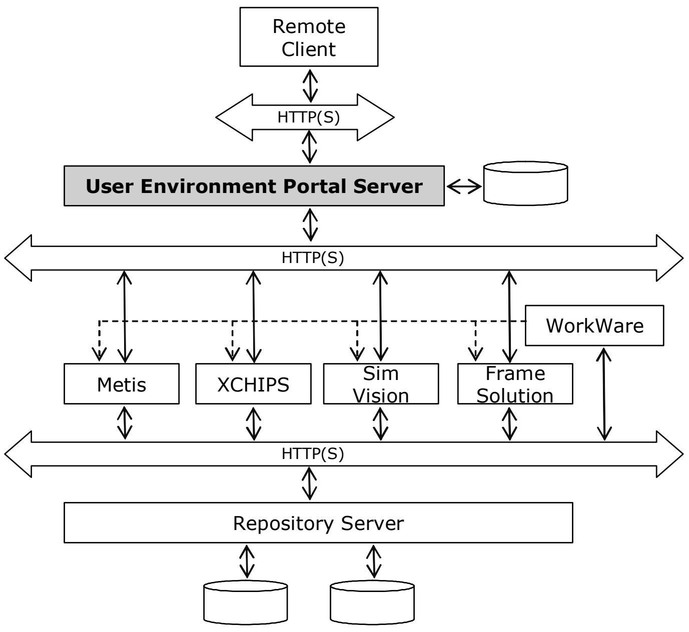


Figure 73. The EXTERNAL Infrastructure [491].

METIS, FRAMESOLUTIONS, UEPS and SIMVISION are commercial tools, while XCHIPS and WORKWARE are research prototypes developed over several years. The interactive process models are the core means of integration. They are stored in a WebDAV repository residing on a web server. For the representation and interchange of models, an XML DTD is defined. METIS is used for building and visualising rich, up-to-date models of the joint project, fostering common understanding and enabling the participants to plan their joint work. XCHIPS has weaker visualisation capabilities, but allow for real-time collaborative modelling and focused collaboration in the context of particular workitems. XCHIPS provides contextual support for synchronous collaboration, and WORKWARE for asynchronous collaboration. FRAMESOLUTIONS automates standard procedures.

The open service model of WORKWARE proved valuable in facilitating control integration. Since all the other tools could be invoked from a webpage, they could easily be modelled as WORKWARE services. Such services may be parameterised, e.g. with values defining which file to open in METIS or which process to execute in FRAMESOLUTIONS. As Figure 73 shows, WORKWARE thus became the glue that allowed other tools, web pages and the UEPS portal to invoke EXTERNAL functionality. In the current version, WORKWARE runs on a separate server, but a re-implementation of WORKWARE as part of a commercial UEPS release has been initiated [491].

7.4.1 Model Data Integration

The infrastructure tools activate process models. However, each tool had its own language and metalanguage, adapted to its specific purpose. A common metamodel called EEML (Extended Enterprise Modelling Language) was therefore designed to enable syntactic and semantic interoperability. EEML follows WORKWARE in most areas:

- Workitems (called *tasks*), decision connectors, flows, resources, resource roles, is-filled-by and candidate relationships form the core of EEML.
- EEML objects and relationships are instances.
- A generic scheme for defining new classes and properties is included.

EEML makes few commitments to specific metametamodel features, since all the tools have different, hardcoded metametamodels. WORKWARE's extensional and intensional classification mechanisms are thus not supported by EEML. Classification is left to each individual tool. The metamodel lays down some rules, but the repository does not actively enforce them. Instead they must be implemented in each system. As the following discussion shows, the tools implement EEML quite differently.

METIS

METIS is an open enterprise modelling tool [71, 309, 311, 312]. It allows user organisations to define their own local languages, but comes with a set of predefined templates. METIS is mainly a tool for articulation and visualisation, not for automated activation. The metametamodel of METIS posed some constraints on EEML. Its basic constructs are object, property and relationship. Relations are binary, and connect objects only. Properties can take simple data values and user-defined enumeration types, but may not refer to other objects.

XCHIPS

XCHIPS [233] is built on top of a generic infrastructure for cooperative hypermedia. Like most hypermedia systems, the basic constructs are nodes (objects) and links (relationships). Links are first class constructs, so both nodes and links are typed and possess properties. Like WORKWARE, but unlike METIS, XCHIPS allows addition of properties to local instances. The system supports *metamodelling by examples*, where new classes are defined by prototype instances. While METIS users typically view or edit the whole model, XCHIPS users work on a small portion at a time. The work breakdown structure is used for dividing the model into manageable chunks. This design is chosen because real-time collaboration across the Internet requires limited data sets to yield acceptable performance.

SIMVISION

SIMVISION [295] simulates the expected course of events in a process model. A wide range of quantitative parameters tunes SIMVISION to different contexts. We also have to articulate the amount of work involved in each task, the percentage of a person's time allocated to the project, the skills of each participant etc. SIMVISION includes some additional constructs like meetings, coordination relationships between concurrent tasks, and reporting hierarchies among persons. These were included in EEML. SIMVISION limits the depth of the work decomposition. No more than two levels can be simulated. This implies that for typical EEML models, the work breakdown structure has to be collapsed prior to simulation.

WORKWARE

The metametamodel in WORKWARE differs from the rest of the tools in that it is not primarily designed for graph visualisation, but for textual viewing and editing. Consequently it does not contain a relationship construct. Instead attribute values can refer to other objects. Conversion between these representations was not always straightforward, due to the tools' different purposes. For instance, reusability and the need for reassigning roles, requires indirect resource allocation, where one resource role is-filled-by another recursively. In WORKWARE, we are in a work performance mode, and mostly concerned with who is filling the role, not the details of resource management. Consequently resource allocation is represented in WORKWARE as attributes that have the role name as their name and the actual resource who is at the end of recursive is-filled-by relations as its value, following the metamodel (Figure 29 on page 95).

7.4.2 User Interface Integration

Since the EXTERNAL tools were originally implemented with different technologies, it was not feasible within the constraints of the project to provide a fully integrated user interface. Instead all tools implemented some means by which they could be started from a web page, and WORKWARE conducted the service invocations.

7.4.3 Integrating Workflow and Real-Time Groupware

XCHIPS provides a wide range of real-time collaboration services. These services are included in the context of particular items of work, the worktops in WORKWARE:

- Chat and NETMEETING for synchronous communication, complementing the asynchronous email integration of WORKWARE.
- Collaborative modelling, where participants jointly manipulate an EEML model.
- Distributed meetings where synchronous collaborative editing, modelling, drawing and gesturing services may be utilised. The resulting artefacts are automatically included as document resources attached to the meeting workitem.
- Various education-assisting features that allow people to experiment, learn, and teach each other, e.g. collaborative model browsing and test-runs through the processes.

- Resource management through a visualisation interface, where tasks are grouped by who performs them, and filtered according to a wide range of criteria.

In total, these services facilitate close collaboration, cooperative management and learning in the process support environment.

7.4.4 Simulation of Ongoing Projects

The core benefit that the infrastructure brings to SIMVISION is the potential for more realistic models. The gap between real and modelled processes is decreased when the models are enacted. Flexibility and ease-of-modelling enables process participants to change models to reflect improved understanding and capture unforeseen events. Currently SIMVISION has weak support for dealing with ongoing processes, but the system can simulate a wide range of scenarios of the same model with different parameters. These features can be utilised manually to tune the simulation model as the work progresses, building on the configurations that have forecasted most accurately so far. This fosters quality and trust in the estimates of the project.

7.4.5 Summary and Assessment

EXTERNAL serves as a proof of concept implementation for loosely coupled interactive process model architectures. It shows that systems with different purposes and underlying metametamodels can interoperate on the level of process models. The simplicity of EEML, for instance achieved through limited classification of constructs, facilitates this integration, not requiring tools to deal with specialised constructs that are not needed for their purpose. EEML has also influenced the Unified Enterprise Modelling (UEML) standardisation project [546].

The purpose of a tool influences its language, meta-language and implementation. An integrated framework for interactive models should thus provide multiple access mechanisms. EEML was a pragmatic design subject to resource constraints. Metametamodel plurality may be required due to such practical limitations, but it caused problems. WORKWARE's metametamodel is designed to meet challenges of interactive models in general. As a further demonstration of its suitability, section 9.2.1 shows how this metametamodel can make enterprise modelling in METIS simpler and more flexible. While the current infrastructure supports data, control and user interface integration, some issues remain to be solved:

- Locking and versioning of models.
- The shared model repository has no change notification service, though WORKWARE and XCHIPS have implemented a protocol for change notifications during enactment
- The granularity of model access is fixed to the file level and references across files are not managed. This scheme is satisfactory for modelling, but not for enactment.
- More modelling constructs should be shared among the tools, and metamodelling should be supported in an integrated manner.
- User interfaces should be integrated, combining components from different tools.
- Services should be better integrated across tools through a standardised interface, e.g. based on web services.

These limitations mainly reflect the current state of the implementation and do not invalidate the quality of the proposed designs. However, in extending the infrastructure to meet remaining challenges and provide a stable commercial product, we could be faced with practical difficulties that require rethinking of the basic designs. The implementation thus backs the claim of feasibility, but it is incomplete.

7.5 The EXTERNAL Modelling Language

Since WORKWARE does not provide a visual modelling interface, the models in this thesis have been built with METIS[1]. The METIS EEML template implements some of the proposals from this thesis, utilising a number of features in the tool. This section discusses strengths, weaknesses and potentials for further improving the EEML template.

7.5.1 Basic Modelling Functionality in METIS

METIS is an open modelling tool. Through metamodelling, users can define new object and relationship types. In order to make the large set of available modelling constructs manageable, related types are grouped in *domains*. EEML includes domains for process, resource and goal modelling. Domains that are used together for a specific purpose are grouped in a *metamodel*. A *template* is a starting point for new models. Templates include a metamodel with domains and possibly some initial content (model objects and relationships). EEML is defined as a template and a metamodel. In the context of a particular model, users may dynamically include new domains that were not originally part of the template metamodel.

7.5.2 Multiple Views

METIS separates modelled data from visualisation. This implies that different selections of model objects can be visualised with different layouts, and that each model element may be shown in a number of different views. Different symbols can be used for the same object in different views. This allows users performing different roles to have their own customised perspective on the model. Models may be edited through any view, and METIS offers a wide range of services for generating new views based on user-defined selection criteria. Rich and complex models can thus be accessed though simplified, personalised and contextualised views.

7.5.3 Methods for Deriving and Computing Properties

METIS also allows property values to be computed from other values in the model. This is achieved though the *method* construct. Methods are implemented in a programming language, packaged in dynamically linked libraries, and access model data through the METIS API. Standard methods exist for extracting values from related objects, also through recursive navigation along modelled relationships, and for basic computation and string manipulation. This scheme allows derived and computed properties, increasing the degree of explicit semantic holism in models. For instance, an EEML resource role has a derived property that points to the object that fills the role, traversing is-filled-by-relations recursively through indirect allocations, similar to the way WORKWARE collapses these structures.

[1] The first implementation of EEML was made by the author in METIS 2.2, while the version used in EXTERNAL is implemented by Dag Karlsen in METIS 3.2.

7.5.4 Macros for Dynamic Visualisation

Users may also draw *symbols* for viewing objects and relationships in METIS. The graphical building blocks of symbols can be named, and their properties (colour, size, visibility etc.) can be controlled dynamically by *macros*. This enables objects with different properties to be visualised differently. Macros were utilised in a number of ways:

- Workitem colours depend on states,
- Connectors and flows change colour when they have been activated,
- Resource properties are utilised in macros to visualise the APM resource modelling vocabulary (Figure 74).

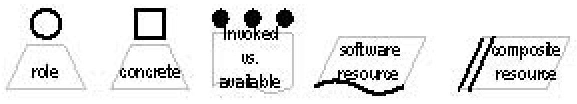


Figure 74. Resource properties that can be visualised with macros.

7.5.5 Semantic Holism in EEML

In some ways EEML violates the principles of semantic holism proposed in Chapter 6. Often, the reason is limitations of the tools' metametamodels. In other cases, EEML simply reflects state of the art modelling approaches. Many of the proposals in Chapter 6 were triggered by discussions concerning the design of EEML. There are two versions of EEML, and the second [286] fixes some of the problems of the first [285]. For instance, different kinds of decisions were originally separated into different classes, but in version 2, properties were used to a greater extent, simplifying the language. In both versions, resource modelling is defined in a rather atomic manner. EEML has separate class hierarchies for *roles* and concrete *resources*. This reflects the need for a clearly defined vocabulary. For interactive models, on the other hand, section 6.4 argues that the difference between abstract roles and concrete resources is one of state and not of substance, and thus should be encoded by a property. EEML thus doubles the number of primitives needed for resource modelling. Since there are many dimensions resources can be classified according to (cf. Figure 74), this weakness should not be ignored.

In addition to simplifying the language, semantic holism makes resource modelling more flexible. If concrete and abstract resources had the same class, one could for instance model resource allocation simply by placing concrete resources in the resource signature of a workitem. In EEML today you must first define and name a resource role, then the concrete resource, and finally draw the is-filled-by relationship between them to accomplish the same. This is illustrated in Figure 75.

Another feature of semantic holism is that it allows the same objects to be used in different contexts, and adapt its meaning to the context. In Chapter 6 we exemplified this feature with person objects in standalone mode as well as workitem, organisation, group and flow contexts. In EEML version 1 roles were only allowed to appear within workitems. During initial use, it was recognised that standalone roles helped make the models more reusable, representing standard project roles independent of workitems, so this constraint was relaxed.

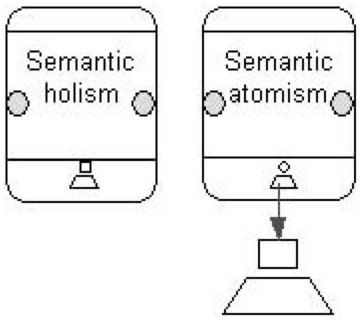


Figure 75. Resource allocation simplified by semantic holism.

7.6 Reuse

The incremental development of WORKWARE started with basic services for information sharing and work management. Then coordination, enactment and modelling were added. Process knowledge management was the third step, and time did not allow all parts of the reuse framework to be implemented. Above the implementation of the metametamodel was described. Policy-controlled inheritance of dynamic changes was not fully implemented. It however represents a generalisation of some specific reuse mechanisms that were implemented in order to meet requirements from EXTERNAL users. This section describes some of these functions.

7.6.1 Templates

Both METIS and WORKWARE support creation of new models based on templates, as well as copy-and-paste reuse. METIS creates new models from templates, while WORKWARE creates new objects. Combined, these tools thus support both initial reuse during the early planning of a new project and ongoing plug-in of templates for specific tasks in the project. There is currently no template management component in the EXTERNAL infrastructure, so users must manually organise their template repositories. As more cases are performed and the set of available templates increase, users do require extended support in this area. Despite these limitations, the case studies in Chapter 8 show that *manual* model reuse does take place.

7.6.2 Classification for Reuse

Classification structures reuse in a number of ways in the current implementation of WORKWARE. As described in section 7.3.1, user interface policies and service configurations can be defined at the class level and inherited along specialisation relationships. When deciding what user interface component to use for a property of an object, the system looks up policies for the particular property inside the given object's class. If no special policy is defined for that property, policies for the type of the property are applied. In both of these cases conventional inheritance is used, so that superclass policies also apply to subclasses. Through metamodelling, users can redefine class hierarchies, add or remove classes etc. Most of the extensional superclasses defined in the basic installation of WORKWARE are motivated by the need for defining common policies for various interactors. General classes like *Process modelling*, *Data management*, *User interface preferences*, *Service management*, and *Objects with event log* exemplify this.

The classification structures are also utilised in local metamodelling. For instance, when a user adds or removes an attribute, she chooses the object instance or one of its classes as the scope of change.

7.6.3 Utilising the Work Breakdown Structure for Reuse

Currently WORKWARE makes little use of work breakdown or other decomposition structures for reuse. The work management interfaces provides links to parent and child resources for each workitem, so that users easily can browse e.g. parent documents from the document folder of the worktop for the child workitem. The access controller, however, utilises inheritance along decomposition structures. This implies that users can define general access rights for a project, which applies to all its tasks, documents etc. Similarly, access rights to objects are inherited by all the properties of the object. Project user groups (*Participants*, *Customer*, *Responsible*) are inherited from the workitems that are part of the project (up the hierarchy).

7.6.4 Reuse along the Flow of Work

The flow relations between decision connectors (and indirectly between workitems) are also activated as navigation links in the user interfaces of WORKWARE. From the document folder of a worktop users may navigate to the workitems that the current item receives flows from or sends flows to. Although navigation mechanisms do not automate reuse, they help participants identify resources that may be of interest to the current work, thus fostering manual reuse.

7.6.5 Resource Allocation and Personalisation

In WORKWARE, the current user's preferences control the user interfaces, not which user is responsible for the item. In the awareness engine, however, person role relations are utilised for reusing event notification filters. When a user invokes the *catch up* service, a time filter that stops all old events, is added to the global set of filters for that user. The next time the user asks to see event notifications, this global profile is reused to all relations that the user has to workitems. Another example is access control policies, which are reused along the relations between workitems and information resources. Through links from the workitem directory to where the documents are stored, workitem access rights are inherited.

7.6.6 Parameterisation and Properties

In section 7.5.3 we saw that methods and relationship traversal schemes were applied to support derived properties in METIS. WORKWARE also implements a few parameterised reuse rules. When users model their processes visually, they seldom bother to name objects like decision connectors and flows. When the same models later is accessed in the textual interface of WORKWARE, however, names are needed. Currently unnamed decision connectors are named after their owner workitem, e.g. "Input to <workitem name>", or "Output from <workitem name>". Flows are named after their target connector. Similarly, user interface policies and service configuration objects are automatically named based on the situation that they apply to; e.g. "Services for <workitem name>" or "Attribute edit style for <class name>s". In some cases values are extracted from the context of the request, e.g. when a user defines a new workitem, she is by default responsible for it.

7.6.7 Project Model Lifecycle Support

There is currently no specialised support for model-driven processes of reuse and harvesting in the EXTERNAL infrastructure. In order to enact meta-processes like the ones defined in section 5.6, general mechanisms are utilised. However, since such meta-processes are quite static and system-oriented, a more automated solution would be preferable, guiding users through the steps of e.g. defining a new project. Specialised interfaces for project definition have thus been built both in XCHIPS and UEPS. These solutions utilise the parameterised services of WORKWARE and other tools to perform tasks like define new project, model the project, start performing the project (import to WORKWARE) etc.

7.7 Summary

This chapter has documented the implementation of the proposed designs in the WORKWARE prototype and the EXTERNAL infrastructure. This implementation work has involved a number of people from software companies, research institutes and user organisations. Any such implementation effort requires pragmatic trade-offs between ideal research objectives and feasible practical solutions. In EXTERNAL, particular emphasis has been put on providing the industrial cases (Chapter 8) with tools that they are willing to put into operation, tools they can use to build customised solutions for their processes. Most of the designs presented in Chapters 4, 5, and 6 have been implemented, except enactment policies, extended decision semantics, user-defined enactment rules, and the reuse policy framework. Detailed designs and usage experience should be sufficient to show that the proposals are implementable. The survey of related work (Chapter 9) also pays particular attention to the feasibility of the remaining implementation work.

Chapter 8
Usage Experience

The EXTERNAL infrastructure has been applied in a number of projects. These cases constitute a representative selection of knowledge intensive virtual enterprises. One is a business consulting firm interacting with its customers. The second is a network of small software companies. The third is an international research project (EXTERNAL itself). Interaction between users and developers has ensured an ongoing practical validation. This process started during the development of WORKWARE in the AIS project [254]. In addition to the EXTERNAL case studies, this chapter also discusses ongoing use of WORKWARE in other settings. In-depth model examples are complemented by formal evaluations with interviews and questionnaires.

8.1 Extended Enterprise Requirements

A virtual enterprise consists of a number of organisations collaborating across a networked IT infrastructure. In EXTERNAL, the term 'extended enterprise' (EE) refers to virtual enterprises integrated by active model-driven infrastructures. This introductory section outlines how interactive modelling can meet the challenges of EE planning, operation, and management. An extended enterprise is typically established ad-hoc to reach certain goals, by uniting forces from several organisations. Process models depicting how to reach these goals are therefore natural EE integrators. A collection of requirements have been extracted from the three cases, pointing to a number of core challenges:

- *Knowledge sharing*: Create and maintain a shared understanding of the scope and purpose of the enterprise, as well as viewpoints on how to fulfil the purpose. Language barriers and cultural differences are among the obstacles here [336]. Hence it is not sufficient to support communication with a technical infrastructure, there is also a need to support communication on a semantic level, through a common terminology, agreed-upon descriptions of work practice, etc.
- *Dynamically networked organisations*: The dynamic nature of an extended enterprise represents technical as well as social challenges. During one EE, replacements, removals and additions to the network might occur. For such a body to operate effectively, it is vital that knowledge about ways of working is readily available to all.
- *Heterogeneous infrastructures*: Different organisations have different IS infrastructures, connected by email and publish-oriented extranets, but seldom by work support environments. In such infrastructures, knowledge sharing is often reduced to information dissemination. Common practice across locations is poorly facilitated.
- *Process knowledge management*: Business processes may be described on several levels of abstraction, from abstract models defining the core process logic, to executable models with all necessary details [254]. Advanced process model management is required to integrate these levels of abstraction.

Interactive EE models centred on process perspectives can help organisations meet these challenges. Such models capture a rich set of relationships between the organisations, people, processes and resources of the virtual enterprise. Through analysis and activation, the models become applied as sources of knowledge, providing the basis for learning. Communication is supported by the infrastructure tools, and the terminology of the modelling language. Joint modelling of the enterprise facilitates common understanding, and the extensibility of the modelling language allows local perspectives to be expressed and utilised. Detailed requirements from EXTERNAL users [465], deal with project planning, management and coordination, work execution, knowledge management, and infrastructure management. This practical input helps to validate the general requirements from the literature survey in Chapter 2.

8.1.1 Planning

For project planning, this support was required:

- Simple, visual articulation of all parts of the project from different perspectives, including scheduling, resource planning, budget, organisation, roles, quality assurance, tasks and work descriptions (cf. requirements R1, R2, R3 in Chapter 2).
- Template models for generic processes, and domain specific building blocks (R9).
- Negotiation and merging of heterogeneous corporate procedures (R13).
- Simulation of ongoing and planned processes (R8).
- Easy navigation in the work breakdown structure (R1, R9).
- Scalable methodology, tools and languages suitable for small, simple projects as well as large and complex (R3, R11).
- Easy retrieval of past project plans, performance and change histories (R5, R9).
- Resource management across multiple projects.
- Adding, replacing and removing partner companies to/from the project (R6).

8.1.2 Management and Coordination

For work management these services were prioritised:

- Easy and not time-consuming project monitoring and progress follow-up.
- Deviation detection and notification, with subscription to specific event types.
- Risk management, identifying, assessing, and managing uncertainties (R8).
- Role management, with role descriptions, skill requirements, competence, role assignment and history of re-assignments (R3).
- Definition and management of shared resources.

8.1.3 Work and Collaboration

The infrastructure must provide the easiest and most rewarding way of performing the work, otherwise other tools will be chosen (R1, R7). Work execution requires:

- Highlighting of urgent and high-priority work.
- Cooperation support such as communication, shared workspaces, collaborative modelling, problem solving, decision support, distributed brainstorming etc.
- Work performance support, e.g. task automation.
- Automated information flow between tasks, also ad-hoc for information that was not originally included in the model.
- Implementation of standard problem solving procedures.

8.1.4 Knowledge Management and Learning

For knowledge management and learning these services were requested:

- Management of continuously accumulated knowledge.
- Access to experts, e.g. modelling experts.
- Metamodelling (R12).
- Cooperative model browsing, enabling teaching and mentoring.
- Separate repository areas for different teams (R9).
- Automatic generation of role-dependent views on visual models (R2).
- Models provided as learning resources, with background and explanations (R11).
- Animation, role-play, and guided tours of enactment scenarios and history (R5).
- Methodologies based on actual project experiences (R8).

8.1.5 Infrastructure Management

Infrastructure management deals with system integration, configuration and customisation, and requires:

- Simple user interface, intuitive and based on accepted standards (R1).
- Intuitive, self-instructed installation and start-up.
- Quick and easy establishment and configuration of a starting point infrastructure for a new project, before the project is completely defined and all partners known (R6).
- Different levels of interoperability, from email and a web portal to rich process integration, allowing increased interoperability as the collaboration matures (R4).
- Different technical infrastructures, both regarding software and hardware. For instance, the IT consulting case is set in a low-bandwidth environment.
- Integration with current infrastructures in the user organisations, e.g. compatibility with industry standards.
- Differentiated access control according to project roles. High security on some of the information. Single logon for all tools in the infrastructure.
- Integrated user environment (portal) to all tools, with integrated help system.

8.2 Case 1: The EXTERNAL Project

EXTERNAL took its own medicine as an early experimentation arena for EE methodology, tools, and infrastructure. Experiences from this arena were fed into the further development process for the benefit of the industrial cases. The project plan was thus articulated in early prototype versions of EEML, and later imported to the work execution environments (WORKWARE and XCHIPS). Due to resource limitations and the instability of an evolving infrastructure, it was decided to put particular emphasis on supporting two typical process examples rather than the whole project:

- *Periodic progress reporting*, a mandatory, routine administrative procedure, where reports are written for each work package each quarter, and then collected and sent to the customer twice a year.
- *Joint project planning*, an ongoing knowledge-intensive activity where work package plans are elaborated. Often, planning takes place after reporting, in order to accommodate deviations and provide more detailed plans for the next period.

In addition to these planned case studies, which were carefully evaluated, ad-hoc utilisation of EXTERNAL tools also took place in some groups in the project. The following subsections summarise lessons learned from these cases, focusing on the aspects most relevant for evaluating the interactive modelling approach, the language, and tool support from WORKWARE. More details are available in [232].

8.2.1 Periodic Progress Reporting

The main activity in this case is quarterly progress reporting (QPR). An excerpt of a model for this process is presented in Figure 76. For each of the nine work packages (WP), the WP manager writes a separate report. The report template and actual report are modelled as information resources to these workitems. The project manager is responsible for coordinating and following up the reporting process. In the model, an optional meeting is included for coordination purposes.

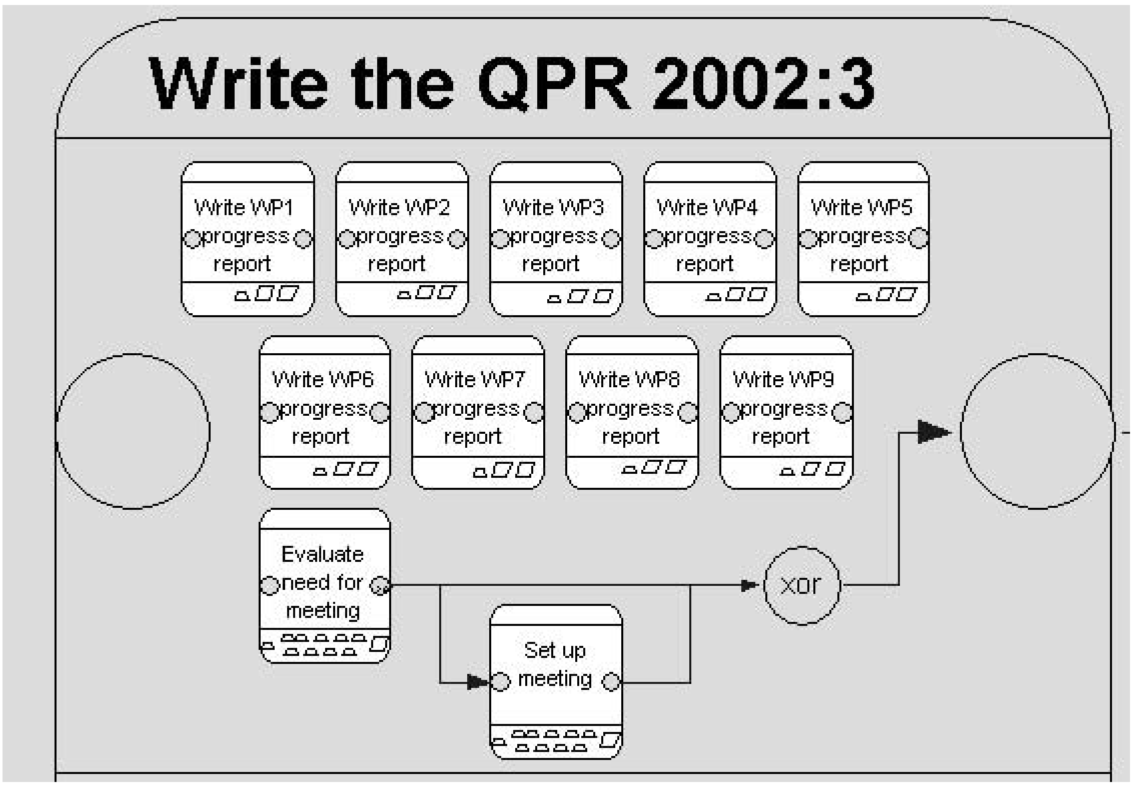

Write the QPR 2002:3
Write WP1 progress report
Write WP2 progress report
Write WP3 progress report
Write WP4 progress report
Write WP5 progress report
Write WP6 progress report
Write WP7 progress report
Write WP8 progress report
Write WP9 progress report
Evaluate need for meeting
Set up meeting
xor

Figure 76. Model of quarterly progress reporting.

In Figure 76 relationships between resource roles have been hidden for readability. Though this process is quite simple, it shows that the interaction perspective helps to limit the complexity of the model. For instance, we need no flows from the start of the main process to the concurrent sub-items. The lack of input flows means that no constraints prevent the items from starting. Another simplification is evident in the location of the workitem "Evaluate need for meeting". This is something that all nine WP managers must do. In systems that only allow one person per workitem, you would thus need nine items. Here all WP managers are allocated to one collaborative item. This allocation is made indirectly through the actor roles on each of the "Write WP progress report" items, so if one of the managers delegates the task to someone else, that person is automatically involved in the meeting as well. (In this example indirect resource allocation does not follow the work breakdown structure). There are more examples where resource allocation captures dependencies that needn't be duplicated to the process dimension. One is the use of information resources to represent the report document parts produced by each WP manager. They are put together in a workitem not shown here, but since the parts are modelled as information resources, we needn't model the information flow as well. The architecture of multiple model interactors, here the document manager in addition to the workflow engine, thus simplifies the models. Another interactive feature utilised in Figure 76, is that the decision whether to have a meeting or not is to be made locally (it is modelled at the output of the evaluation workitem). If the project manager wanted more control, she could, as responsible for the whole process, have moved the decision out of the sub-item. Now any participant can make the decision that a meeting is required, completing the evaluation item and triggering "Set up meeting".

The tasks of the project manager (PM), are supported by the services of the infrastructure, and need thus not be articulated in detail. WORKWARE worklists provide overviews of the current state of the process, helping the PM to see which WP managers have not yet written their report. Through the *Mail to all* service on the worktop, the project manager sends reminders to the WP managers when it is time to write a new report, and again when the deadline approaches. The PM role was however reassigned five times throughout the project, so there was need for explicit coordination routines that the new manager could reuse. This was modelled as a process around the core work (Figure 76), as depicted in Figure 77.

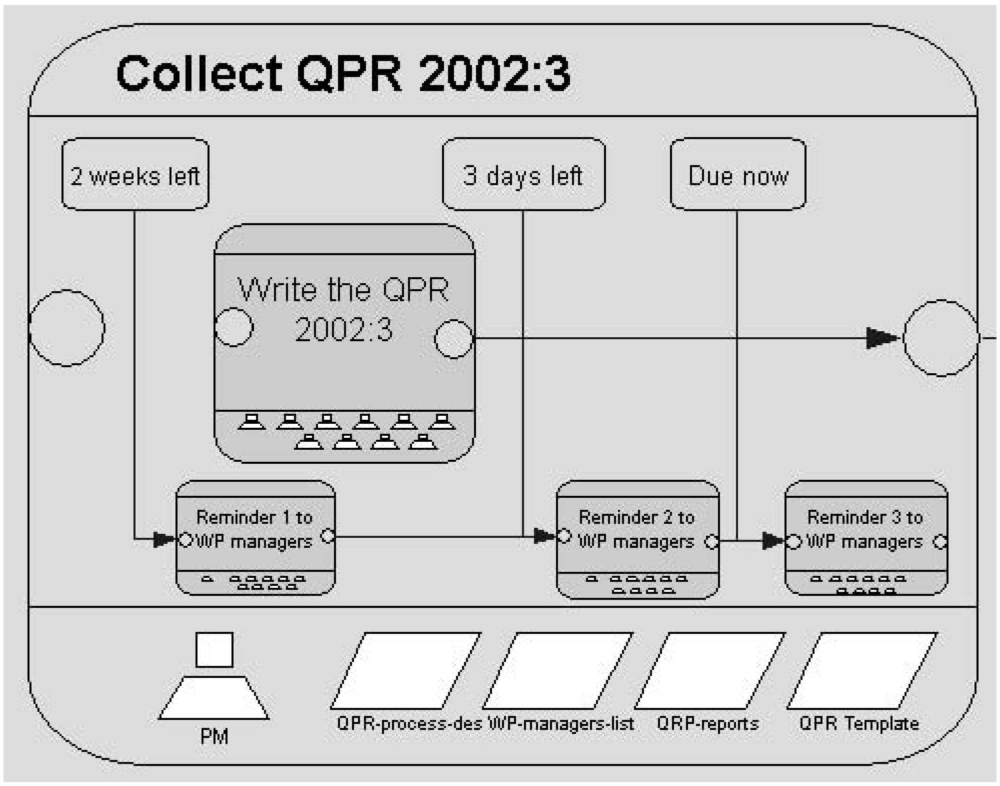


Figure 77. Management procedures in quarterly progress reporting.

Progress reporting is a routine, administrative procedure that recurs throughout the project at regular time intervals. This model was thus reused a number of times. When the process was first articulated, the support for reuse was limited to copy-and-paste in

METIS. Since the infrastructure was not ready, it was not used for the first reporting period. A lot of the initial learning and alignment of reporting practices across organisations and countries was thus already captured in the first version. However, an updated procedure was implemented a year later, taking into account experiences with working together as well as increased understanding of the capabilities of the model-driven infrastructure. The new version

- Changed the work breakdown structure to make individual responsibilities clearer,
- Made resource allocation more explicit in order to handle re-assignment better,
- Added output flows so that "Write QPR" automatically finishes when all of its sub-items have been completed, and
- Added previous reports from each WP as resources for the "Write WP progress report" items.

Another occasion of end user innovation in the reporting case involved metamodelling. The process in general, and the management work in particular, is time-driven. As shown in Figure 77, the participants decided to model *timers*, a decision connector subclass not part of the EEML at that time, but one that they needed in order to handle exceptions (delays) and coordination. As WORKWARE did not support timers, the project manager herself had to remember to do these things, but at least she was reminded by the presence of the objects in the model. The timers were thus manually activated. For process knowledge management and IS evolution, this information was highly relevant, as it pointed to requirements that were not expressed in the specification documents [465], but emerged during use. The reuse framework presented in this thesis could improve such process knowledge management in a number of ways, including

- Local modifications and metamodelling (adding a trigger-time property to the decisions) supported by the instance-oriented metametamodel.
- Propagation of dynamic change so that updated definitions are used in all instances.
- Parameterisation of model properties. Most of these workitems' names refer to WPs or the current time period, and could easily be generated from parameterisation rules.
- A specialised, semi-automated reuse metaprocess called "Create new periodic progress report" could be included as a service in WORKWARE. Based on some property values from the user, e.g. the name and deadline of the current period, as well as the current project plan (showing what work packages exists and who their manager is), the reporting process model could be automatically generated.

End users' evaluations of the reporting application are discussed below.

8.2.2 Joint Project Planning

Project planning was selected as the second case from the EXTERNAL project because its characteristics complemented the reporting case. Planning is a more knowledge-intensive, ad-hoc activity, and it utilises modelling tools for work performance. While the emphasis of reporting was activation and reuse, planning primarily concerns model articulation. It was also expected that the need for coordination between different work packages would require the collaborative modelling services of XCHIPS. The first implementation included the *plan* (a process model) as well as the *planning process* (metaprocess), but not the operation of the plans. WORKWARE was not to be used.

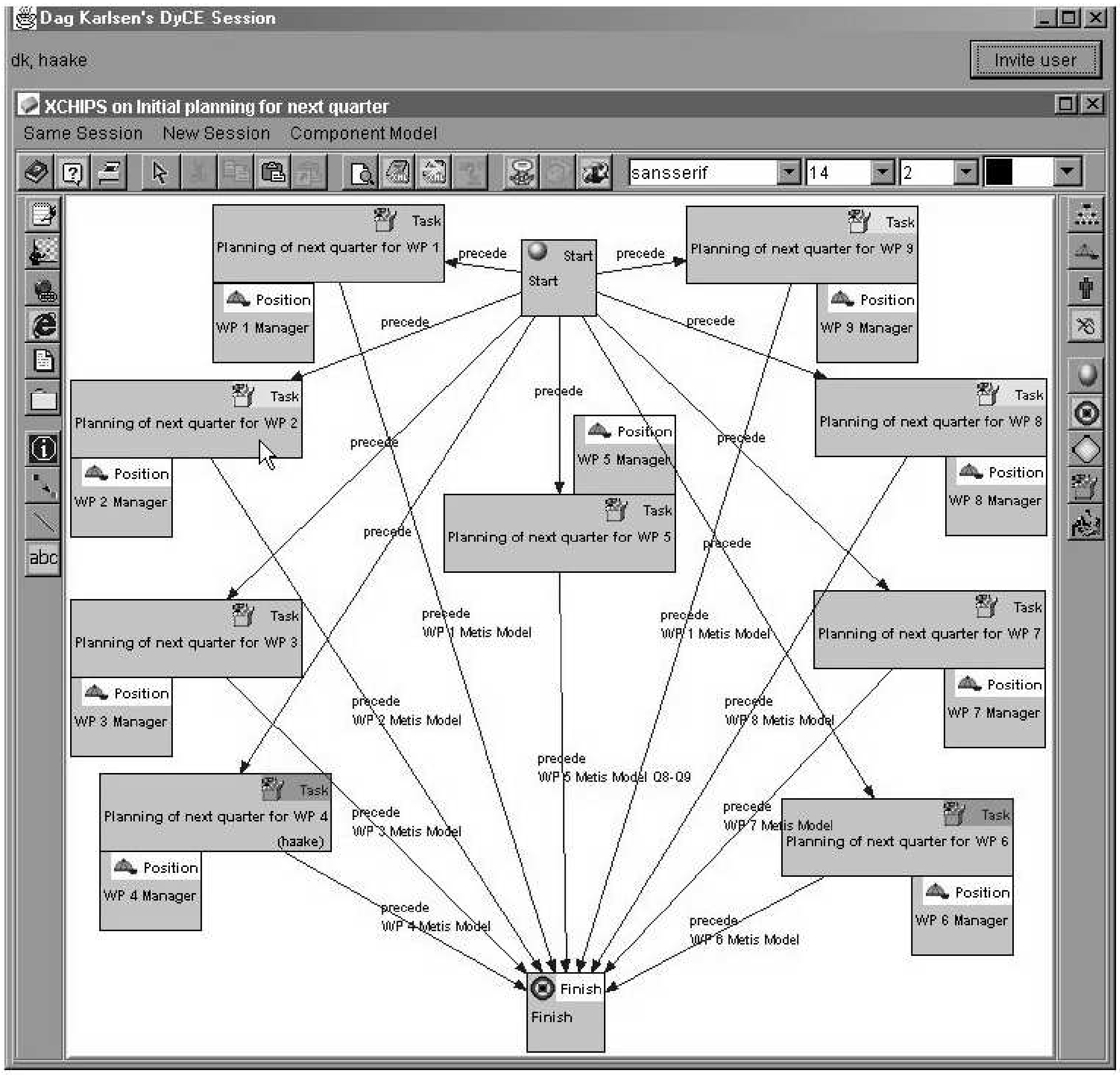


Figure 78. Model of joint project planning in XCHIPS.

The planning process was modelled in EEML and enacted in XCHIPS. An excerpt of the model is shown in Figure 78. XCHIPS supports closer collaboration than WORKWARE. When two people work on the same item, they immediately see the effects of each others' actions. The interface provides real-time awareness of who is currently working (*dk* and *haake* in Figure 78), and shows the current status of the tasks by colour coding (as in METIS and WORKWARE). The use case report contains an example of how these features were utilised for defining a template [232]:

> *"Once the joint planning (JPL) process model was finished, one designer created a work package model template in the METIS modelling environment and made the template available by using the shared repository"* [...] *"Subsequently, she put a link to the template into the JPL process model. Now, another designer used that template to create a sample work package model for WP 4, by using modelling services. This model was reviewed by the first designer and improved during a number of iterations. The final example model was made available in the shared repository and linked to from the JPL process model. This mixture of largely*

asynchronous work and some synchronous discussions was greatly facilitated by the shared repository, collaboration, and modelling."

The template produced here is typical. It includes a basic structure for objects, with separate folders for tasks, inputs, outputs, organisations and people, as well as a project document archive. Some elements, e.g. parts of the archive and the organisational structure, are shared among the work packages. The inputs to one WP in many cases are the outputs of another.

This example shows how (meta)process support can facilitate knowledge management. XCHIPS was also used for enacting the process of *defining new projects* in this version of the infrastructure, invoking METIS to let users define the first plan of the project and then forwarding it to WORKWARE. However, real-time collaboration met technical difficulties with firewalls and limited bandwidth across the Internet to the project partner in Greece. The full collaboration infrastructure was not as easy to start using as e.g. WORKWARE. Consequently, for version 2 of the infrastructure, a web-based solution replaced XCHIPS for project definition.

8.2.3 Evaluation Results

The QPR and JPL cases were subject to a formal evaluation where 10 people answered a questionnaire [105, 287, 312, 431]. The author of this thesis was not involved in this evaluation. It is thus partially independent. The same questions were asked after the first period, when none of the EXTERNAL tools had been used, and then again after the second period, during which the infrastructure had been installed. For the reporting case, time spent, perceived quality of results, and the need for outside help or documents, showed great improvement [431]. Part of this improvement could be due to learning that would occur anyway from the first to the second cycle. However, a baseline survey of the similar process of "Summary Cost Statements", showed less improvement than QPR. Some participants proposed to use WORKWARE for future cost statements as well.

For the planning case, opinions were more mixed. Some of the respondents felt quality and effectiveness had improved, while others claimed the opposite. A clear majority however thought the plans had become more accurate. When asked what was the most important problem in planning, half the respondents originally said lack of collaboration. After having tried the tools, however, all but one chose "identify dangerous delays". It was also reported that *"initial experience shows that the current EE Infrastructure and Tools are too rigid"* [232]. While the numbers from this survey clearly are too small to draw scientific conclusions, the relative results of the two cases, and also for different criteria, are interesting. The opinions by the participants were more clearly articulated (both positive and negative) after tools were applied. Apparently real-time cooperation was not as important as we thought, while simple enactment support seemed more useful. For the further experimentation, it was thus decided to add more work performance orientation to the planning case as well. Experiences from this endeavour are reported below.

In addition to this detailed evaluation of the two case processes, a larger study of the whole project involved another questionnaire complemented with interviews of key participants. This study was carried out by Jarle Hildrum [287, 312]. Again questionnaires were sent out before and after the introduction of the infrastructure. A literature survey highlighted technical viability, cost effectiveness, functionality, and impact on business processes, as key dimensions for this evaluation. Ease of use, easy access to tools, effective communication, learning, and trust were regarded as critical enablers for these objectives. A number of statements were used as indicators for how well each issue was handled, and users were asked to rate to which degree they agreed to the statements. Table 10 summarises the results from the questionnaires.

	Number of people who agree		Number of people who disagree	
Criteria	Before	After	Before	After
Access	2,8	9,3	13	5,6
Useability	-	8,7	-	6,6
Communication	6	12	10	4
Learning	2,4	8,8	13	7
Trust	8	12,8	8	3

Table 10. Summary of evaluation results (average scores) [287].

There are a number of limitations related to these results, including potential bias (respondents were participants in EXTERNAL), and lack of control group. Still, a number of lessons can be learned from the rating of statements and interviews. For instance, customisation was the area within usability that had the lowest score, but the respondents were polarised on this issue. Most people said they "agree a little" that tailoring is easy, while some "strongly disagree". This may reflect the fact that the customisation services had not been made available to all participants, as no specialised customisation user interface exists (other than general editing of policy data objects). Feelings were also mixed regarding simplicity and ease of use, but here the interviews uncovered that while web-based tools (e.g. WORKWARE) were rated high, some other tools were not. Within communication and coordination, WORKWARE-related statements concerning overview of tasks and feedback to information showed the greatest improvements, while real-time communication seemed not to meet the expectations.

As an early assessment of EXTERNAL, a *learning history* [37, 426] was written by a social anthropologist based on interviews with the project participants. The objective of a learning history is to capture a wider multitude of perspectives and subjective views than formal evaluations can. The story clearly demonstrates that different people held widely different views regarding key aspects of the project, e.g. what were the driving forces behind its initiation, what were the key concepts, and what was the meaning of key terms. The experiences of the researcher as an outsider to the project group was valuable in uncovering a local tribal language being developed in the project. Sharing this language was important for team spirit. This story indicates that modelling language extensibility, e.g. through metamodelling, has a key role if interactive models are to facilitate the formation of effective communities of practice in virtual enterprises. Although the general criteria investigated in this study match well the concerns from Chapter 2 of this thesis (learning, communication, trust), the statements seldom were detailed enough to assess individual design ideas. More detailed discussions about actual usage are thus needed to complete the usability evaluation. Some examples were provided above, and more follow in the rest of this chapter.

8.2.4 Action Lists - Emergent Project Planning

The first implementation of the joint planning process took a top-down perspective, where managers were responsible for planning the work inside their work package. Such plans, however, seldom are detailed enough to cover all the tasks that are to be performed. Consequently, the EXTERNAL project also had a web-based action list located at the project web server. This solution had a number of limitations, typical of publish-oriented web environments:

- Only the project manager could change the list, update status, add new actions etc.,
- The actions lacked context, and were often hard to comprehend,
- The actions were not explicitly connected to project plans,
- Actions were not linked to a work environment, documents or tools,
- Although the list could be sorted on different attributes and filtered according to certain criteria (e.g. one list for each person), it was not possibly to add new criteria.

The action lists were consequently not actively used by many of the project participants. During the spring of 2002, it was thus decided to replace them with the EXTERNAL infrastructure. WORKWARE had the central role in this application, managing the actions as workitems. It took just a few hours of work to customise a WORKWARE installation for action lists. The WORKWARE Explorer menu for this case was shown in Figure 72 on page 169. It organises actions according to these criteria:

- Status, e.g. most lists contain only ready and/or ongoing actions,
- Delay,
- Work packages,
- Teams that are responsible for coordinating interrelated tasks across work packages,
- Persons and roles, separating the actions which the current user is responsible for from the ones where she is just a participant,
- Follow-up lists, containing all tasks that the current user is customer of. Lack of follow-up was reported as a major problem with the previous system.

The increased access to edit actions should make the list more up to date. Although the structure for the actions was not connected to a full project plan, teams and work packages provided increased context for the work. Explicit assignment of follow-up responsibility and the ability to look in the event log to see who created the action, made each item easier to understand. The old, static action lists contained 288 actions after two and a half years of operation, while WORKWARE contained 131 after just two months, even though it was installed during the summer holidays. It thus seems safe to claim that the second application was experienced as an improvement by the users. After the action lists had been available in WORKWARE for a while, however, usage frequency dropped significantly. This happened although a general consensus was articulated that the application was useful and should be utilised. A number of factors may have contributed to this decline:

- Lack of management commitment and contractual obligations to use the system.
- Since WORKWARE allows everyone to define new tasks and themselves mark their actions as finished, the project manager no longer had to put these updates into the system himself. Although this relieved him of some duties, it also gave him less responsibility for following up all the actions. For instance, at project meetings, nobody was assigned the responsibility of recording the new actions.
- A number of major deliverables were completed, e.g. final versions of the tools, infrastructure, and methodology. Several of the most eager users thus no longer par-

ticipated actively in the project.

- There were technological limitations, e.g. cumbersome document upload. User interfaces and enactment policies for tasks in general, are perhaps too complicated for simple actions. Instability and poor performance of servers may also have discouraged some users. Performance suffered when the action model grew large.
- For a number of situations, email still remained the simplest and most used communication and coordination tool. In spite of its web and email integration, some may see WORKWARE as yet another tool adding to a complex user environment.

During the main period of use, however, it was noted on a number of occasions that people sent out emails referring to tasks, and pointing to documents uploaded to WORKWARE. This did not occur with the previous application. Also, the fact that project participants chose to use the system in a later project, the writing of the MAESTRO proposal, also suggest that it solved real problems. When the users made demonstrators (videos and online applications) at the end of the project, WORKWARE played a role in six out of seven demos, while the other tools were used in one or two cases each.

This case shows how quickly and easily WORKWARE can be customised to a particular usage need by defining an overall process model (in this case the WP structure), a menu structure, and some specialised worklists and services. After people started to use the application, further customisation was made based on their experience. The case also shows how bottom-up, emergent process articulation can complement top-down project planning and give the organisation a truer picture of what is really going on in the project.

8.2.5 Ad-hoc Applications in the EXTERNAL Project

In addition to reporting and action lists, WORKWARE has been used opportunistically in a few other activities in EXTERNAL. Dissemination activities, especially related to publication and conference participation, were planned and coordinated. During the first full installation of the infrastructure, WORKWARE was used for documenting the installation process. Some installation tasks were planned in advance, but most emerged as response to unforeseen difficulties. Short comments about these problems and how they were fixed were captured as properties of the tasks, while installation guides, readme-files etc. were uploaded as documents. The installation consisted of a number of different tools, and in order to make them work together, several concurrent discussions were carried out, mainly on email. Using the Outlook Web Access mail client integrated as a service in WORKWARE, we could store email messages as HTML files, and include them as documents resources on the tasks they referred to. Once the installation was complete, the WORKWARE server contained a knowledge base that was highly relevant for refining installation processes and guidelines, as well as for diagnosing and fixing errors on different client and server platforms. This case demonstrates that for knowledge management, the system may actually have an important role even if only one person is using it to articulate his work.

WORKWARE is also used in other projects. The document archive and email links that each worktop includes seem to be the initial motivation for using the system. But we have also noticed that people use the hierarchical ordering of workitems, so the basic functionality can serve as a door opener for the workflow functionality as well. The results from the QPR case support this expectation. Feedback from users indicates that for some people, especially in the initial phases, process models are still too complicated. Here it is important that the system can be used without any graphical models at all, by just defining items through web-based work management forms. Although this textual interaction enables you to represent the work in a hierarchical structure with people, milestones, and documents assigned, it does not seem reasonable to model flow relationships this way. But for establishing an initial lo-fi collaboration infrastructure, it is sufficient. As mentioned above, this is a key requirement from EXTERNAL's users.

8.3 Case 2: The Business Consulting Project Cycle

The business consulting case involved primary users outside of the EXTERNAL project. The company in question was supported by process modelling experts from one of the EXTERNAL partners. The company had already defined a procedure for how their projects should be executed. This procedure was available on the corporate Intranet, in the form of textual descriptions and informal visualisations. One of the first tasks for the EXTERNAL consultants was thus to model this procedure, known as the *project cycle*, in the EEML language. Local requirements were then collected, and a customised version of the EXTERNAL infrastructure was installed. The users in this case were novices with respect to process modelling and groupware systems, so they selected WORKWARE as their primary tool.

8.3.1 Reuse of Project Templates

The top-level process in the project cycle template is shown in Figure 79. In addition to the process model, the template also includes an organisational model with typical project roles, as well as the firm's tools, information repositories and document templates. The template contains optional items, that are only needed for certain types of projects, e.g. those with a budget larger than a certain amount. These options are currently modelled as normal decisions. However, since many decisions can be made at project start-up, modelling them as reuse decisions, would simplify the local models (cf. Figure 61 on page 141). Many of these decisions are controlled by properties of the project, so the potential for automated reuse decisions is substantial.

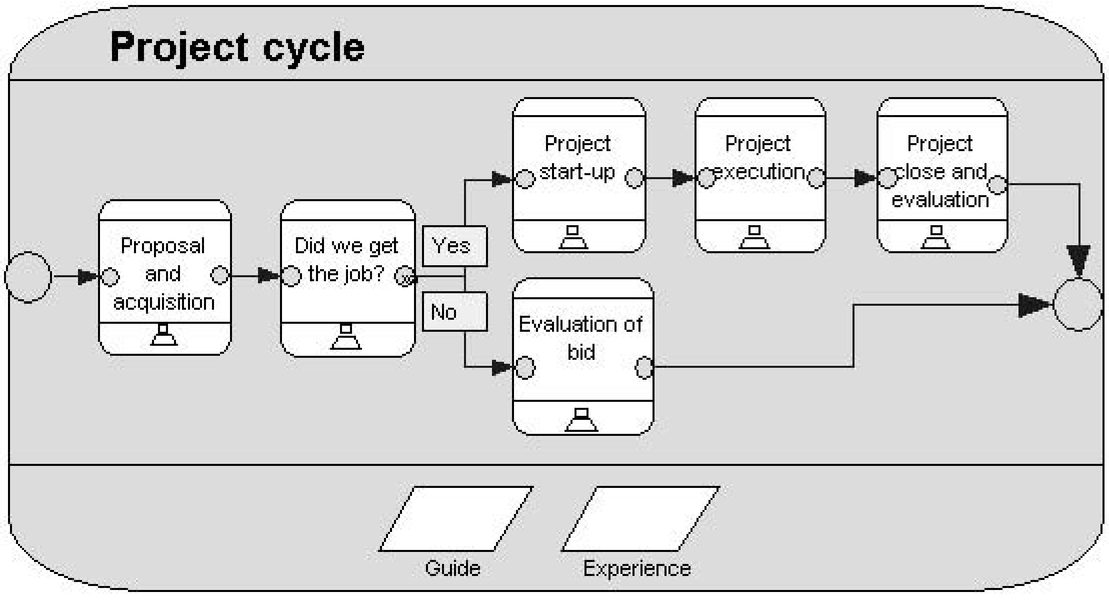


Figure 79. Top level of the project cycle template process.

It is interesting to note that the project cycle mainly defines the administrative work. The actual performance of the project is to be included inside the item "Project work", a sub-item of "Project execution" at level three in the work breakdown structure. This pattern can be expected in a model, which represent management perspectives rather than work perspectives, a typical bias in process modelling. It also reflects the fact that administrative procedures are easy to define and reuse without change across all projects, while the core work is more situated. However, if knowledge management and process improvement are to really create a competitive advantage for the company, the core work must be targeted as well.

8.3.2 Security and Access Control

Improved security and multi-level access control was an absolute requirement from the business consulting company. This was the main reason why access control was prioritised for implementation in WORKWARE. A typical project in this company requires these default access rights:

- Only internal participants should be allowed to read and update all documents,
- Employees not working on a project may not have access to project information,
- Only the project manager should be allowed to grant access rights,
- Participants and customers from other organisations should be allowed to read and change documents and plans within their part of the project, but not the others. In some cases different customers in the same project should not even know about each other. Different customers may have partially conflicting agendas, leading to less-than-full disclosure of information.

The access control interactor of WORKWARE allowed these policies to be articulated at the general level and reused across projects.

8.3.3 Experiences and Evaluation Results

Based on his previous experience with Internet tools, the pilot user in this case regarded WORKWARE primarily as a document repository. The concepts of enactment, work management, and status reporting was not useful to him because in the first project, he was the only participant. Consequently, the system was regarded as too complex and cumbersome to use. This initial reaction indicates that simpler user interface components and enactment policies should be the default for novice users. Though some simplifications were made as part of the customisation process for this case, they were insufficient. The EXTERNAL process modellers were able to reconstruct the project cycle template using the available constructs in EEML. In some cases, however, limitations of the tools and errors in the documentation prevented them from achieving what they wanted. One example was the modelling of template actor roles. The documentation, based on atomic semantics, said that resource roles could only be modelled inside workitems, but they wanted to model the roles independently. When this confusion was cleared up and the semantic holism of the modelling language described, the template was adjusted.

8.4 Case 3: IT Consulting in an SME Network

The final case study in EXTERNAL aimed to support a network of small and medium-sized IT companies located in different countries, mainly in eastern and southern

Europe. Many of these companies are owned by the same group, and have cooperated on a number of projects. Three cases with different characteristics were selected [180]:

1. *Proposal submission for government funding,* a simple and well-defined procedure.
2. *Software development subcontracting,* a case of medium complexity.
3. *Management of a LeonardoDaVinci project*, a complex and unstructured activity.

An overview of the characteristics of these scenarios is presented in the table below.

Property	Proposal submission	Software sub-contracting	Project management
Main objective	Flexibility	Maintainability, reliability	Reliability, adaptability
Duration	Single unit	Long term alliance	Temporal
Topology	Fixed structure	Dynamic	Mixed
Participation	Single alliance	Multiple alliances	Multiple alliances
Coordination	Tree structure	Tree structure	Star structure
Visibility	Single level	Multiple levels	Multiple levels
Collaboration	Activity coordination	Distributed process management	Joint resource management, co-supervision

Table 11. Characteristics of different SME network scenarios [180].

8.4.1 Process Diversity and Model Diversity

It is interesting to see how these differences manifest themselves in the process models. Table 12 shows the number of primary objects of each category in the models of the three cases in this study. For the first two cases, we clearly see that the increased complexity of the cases is reflected in the size of the models. The project management case, however, has a rather simple model. The reason for this is partly that more work has been devoted to studying the two simpler cases, but it may also reflect that project management is harder to articulate than administrative work. For case 3, just the management activities were articulated, and not the core work.

Following the history of these cases, it was interesting to note that software subcontracting, the most elaborate case, originally was modelled as a copy of the project cycle from the business consulting use case (Figure 79). This template was generic enough to be transported to another country and application domain. The fact that the participants in the SME networks had limited previous experience with process modelling also helps to explain why they would rather start with a template than from scratch. Over a couple of months, however, the software subcontracting model evolved, new items were added to all levels of the work breakdown structure, and existing items were renamed to fit the local terminology. Here we saw the process of template *appropriation* in action.

The project management case was modelled as two separate processes, one for the work before the project actually started, and another for the management activities to be carried out during the project work. This modularization makes it easier to reuse the latter process, as management is an ongoing activity that recurs many times throughout the lifecycle of the project.

Property	Funding proposal	Software subcontracting	Project management
Number of workitems	25	80	10
Depth of work breakdown	3	4	1
Number of actor roles	4	25	9
Number of object/tool roles[2]	0	19	18

Table 12. Statistics for models of different SME network scenarios.

8.5 Final Evaluation Results

One year after the survey discussed in section 8.2.3, all the three EXTERNAL cases were subjected to a joint evaluation [105]. A questionnaire was sent by email to 19 users, including managers and project participants. They were asked to rate how much they agreed to statements (both positive and negative) on a 7 point Likert scale. Jarle Hildrum also carried out in-depth interviews of some of the participants.

Frequency of use, user-friendliness and the usefulness of provided functionality were assessed. In general, inexperienced users responded neutrally to all categories of questions. People who had used the tools, were typically slight positive, giving average ratings between 4.8 and 5.6, where 4 is neutral and 7 is maximum. It was thus suggested that "*the user threshold of EXTERNAL is not prohibitively high*" [105].

There were some interesting differences among the responses. Figure 80 shows that WORKWARE was by far the most widely used tool. The evaluation report states, *"it is paradoxical that the tool most widely used is a prototype while the tool that has been used to the least extent is a globally available commercial tool"* [105]. Asked why they did not use the other tools, most people responded either that the tools were not relevant for their work, that they had never learned to use the tool, or that they prefer to use tools that they are more familiar with. Poor integration among the tools, especially at the user interface, was also reported as an important shortcoming. One respondent said, "*the only aspect of tool integration that he had experienced was accessing WORKWARE on the Internet*" [105]. The problem of reaching critical mass, that the people you collaborate with must also use the tools, was also highlighted, and in particular the need for a repository of useful templates. Our emphasis on motivating input to the repository by end users thus seems justified.

Among user-friendliness issues, the highest score (5.8) was awarded to the ease with which tools could be updated according to changes that take place in the project. 15 out of 19 informants agreed that the security of the infrastructure was satisfactory, indicating that the access control interactor fulfils requirements. But there were also negative remarks. *"A manager from one of the case study companies reported that he once tried to convince a group of employees in his company to use WORKWARE to support collaboration in a distributed project, but that the attempt failed because the employees simply did not have the time to learn how to use the tool"* [105]. The following section shows that such opportunistic use of WORKWARE has occurred in other settings, so this clearly is not just a technical problem.

[2] This number refers to the roles modelled as separate objects. In addition, all cases included roles on each task.

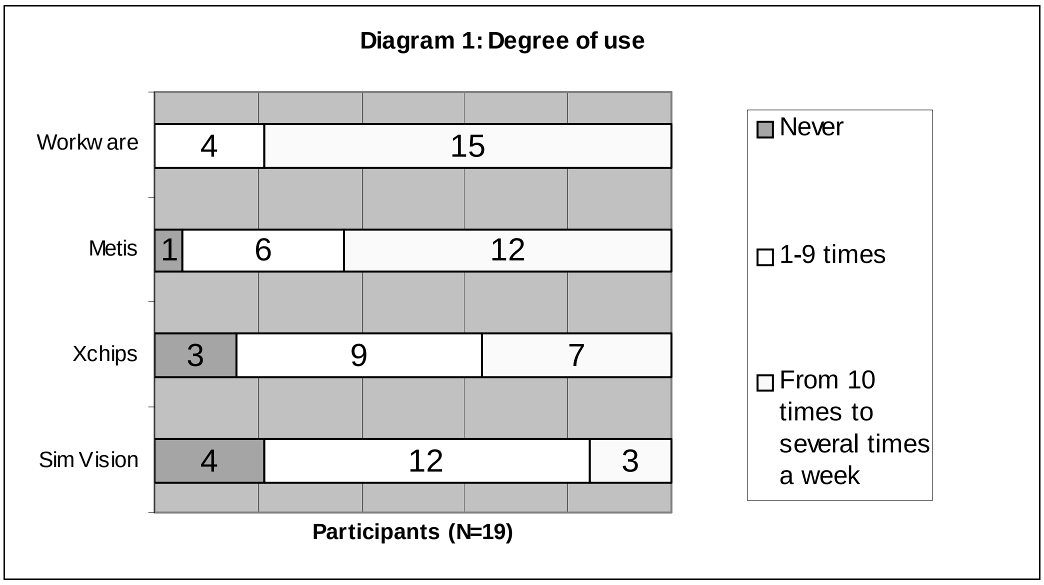


Figure 80. Frequency of use for E*XTERNAL tools [105].*

More disturbingly, an inexperienced user from the business consulting case reported that he had problems with the logic of WORKWARE: "*Every time I log on, I spend some time trying to remember how to use the system. I keep asking myself what the logic is because it is not self-explanatory. For us, the main purpose of the tools is guiding people through project work, but this is difficult because the graphical interface and the use of symbols are not intuitively easy to understand. This can probably be helped without incurring very high costs.*" [105]. System appropriation, involving users in customising terminology, structures, visual interfaces and processes, must thus be paid more attention to. As this user grasped, WORKWARE's capabilities for customisation, configuration and extension has a potential for bridging the gap between local domains and system logic.

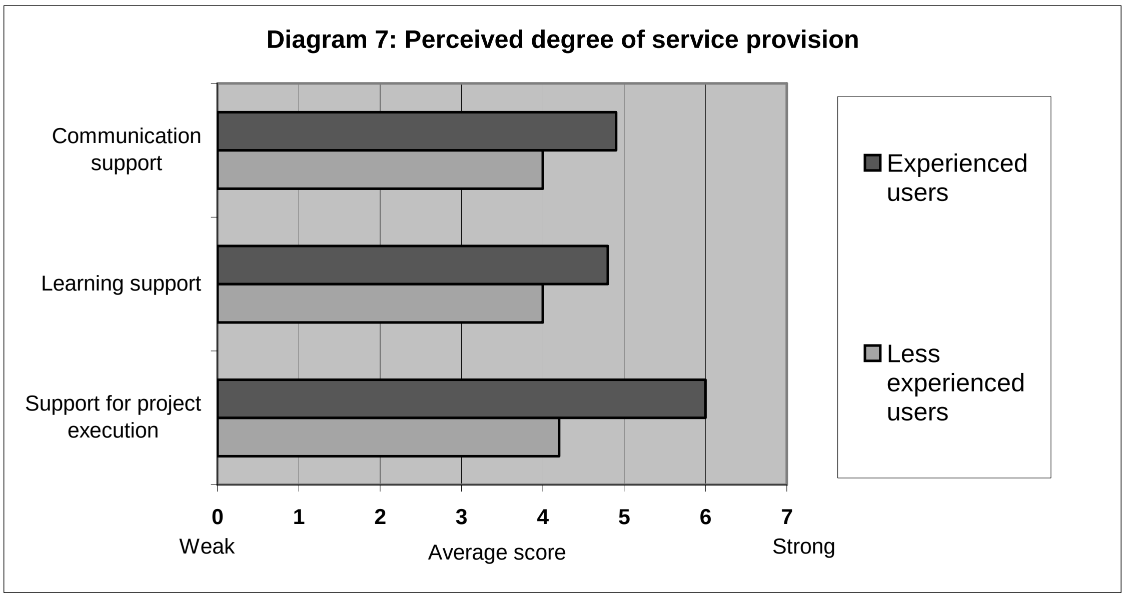


Figure 81. Perceived degree of service provision [105].

When assessing the functionality offered by the system, users who had some prior experience appreciated WORKWARE's execution environment (cf. Figure 81). One respondent noted that "*WORKWARE is a good tool for mapping project work, but that it should be complemented with a functionality that notifies users when several people are working on the same document*" [105]. Others noted that the tool should be more closely integrated with email, while one participant required support for financial aspects of project management. After having experimented with the tool, users thus contribute with clear requirements for addition of more interactors.

In general the hypotheses that task management with simple, interactive enactment services is useful seems to be backed by the evaluation results. Comparatively favourable results have been achieved in spite of technical problems with firewalls that prevented access, poor performance and cumbersome integration. In many cases, the organisational and management commitment to use the infrastructure was limited. It is however also clear that more long-term field studies with a stable infrastructure are needed.

8.6 Other Experiences

In addition to the case studies in EXTERNAL, other projects have used WORKWARE as well. This usage is largely opportunistic and experimental, and has not been supported by any organisation.

- *Advanced Intranet Cooperation* (AIS) was the project where WORKWARE was originally conceptualised, designed and implemented. Users participated in ongoing requirement specification, and experimented with the tool in some simple tasks.
- *Learning in the workplace* (LAP) is a Norwegian research project that involves a number of research institutions, software companies, service providers, and users. A customised version of WORKWARE was used in this project. The menu for this application was shown in Figure 72 on page 169. Since the project had limited funding for its first year, most of the work was done in meetings. Consequently, meetings were added as a new class (specialisation of workitem) for this case.
- MAESTRO is a project proposal that builds on EXTERNAL. WORKWARE was used to coordinate discussions, marketing, and exploration of related projects.
- MONESA is a Norwegian research project where partners from AIS and EXTERNAL (Computas, DNV and SINTEF) continue the collaboration. In MONESA, international quality certification by DNV, and the IT department of the Norwegian Defence will be supported by interactive process models.
- Students at the Norwegian University of Science and Technology (NTNU) in Trondheim, whose supervisors have been located in Oslo, have also used the system to articulate plans for their projects and share drafts of their reports with the supervisors.
- The implementation of EEML in METIS has been used in student exercises in two courses at NTNU. More than 200 students have participated in these exercises, modelling 20 different processes [356].

These cases demonstrate interest in the concepts of interactive process modelling from different domains. The continued active participation by industrial partners (DNV and Computas) since 1996 testify to the practical relevance of the approach. Recently, however, some of these projects have stopped using WORKWARE actively. In part, this is due to technical difficulties, but it also seems there is a minimum degree of collaboration needed to sustain system usage.

8.7 Summary

This chapter has summarised experiences and evaluation related to the application of WORKWARE and the EXTERNAL infrastructure. Although not all attempts have been completely successful, the cases indicate:

- That the requirements, social and organisational issues discussed in Chapter 2 are important and recognised in real world cases.
- General usability of interactive process modelling. According to the evaluation, WORKWARE seems to have performed better than other tools.
- Usability of the process modelling language. Users have been capable of articulating rather large and complex models, and examples have illustrated the practical usefulness of semantic holism and instance modelling.
- The enactment support and other model interactors, e.g. work management, awareness and access control, meet requirements and were appreciated by end users.
- Model templates have been created, reused, appropriated and adapted.
- Metamodelling has also been utilised to enable more specialised support, and the customisation services have been applied to construct tailored applications in just a few hours.

Combined, the cases have shown practical use of most of the contributions presented in Chapters 4, 5, and 6. Process knowledge management has been demonstrated in the cases, even though the automated support is limited in the current infrastructure. The full potential for organisational learning anchored in practice can only be realised when a significant number of processes have been articulated and performed, and reusable local innovations have been harvested. More long-term field studies are thus needed. The cases have also uncovered a number of limitations in the current implementation. Although most people seem to agree that WORKWARE is simple to use, even simpler enactment policies and user interfaces should be defined. It is also evident from some of the cases that social and organisational factors heavily influence the success of collaboration and knowledge management tools. These lessons motivate the emphasis put on such factors in this thesis, particularly reflected in Chapter 2.

Chapter 9
Related Work

In addition to the practical validation of the preceding chapters, comparisons with existing technologies are presented here. This analysis shows the originality and relevance of this work for research areas such as workflow management, tailorable groupware, knowledge management, and conceptual modelling. The main hypothesis is that the techniques of instance modelling, semantic holism, explicit decisions, and multiple model interactors yield significant benefits compared to the state of the art. Throughout the analysis, limitations are identified that demand further research and development.

The first section compares WORKWARE's modelling language to existing process modelling frameworks. We then analyse the application of interactive modelling principles to UML and enterprise modelling. These analyses show that the modelling principles in general enable simpler and more flexible articulation. Section 9.3 compares WORKWARE's activation mechanisms to the most relevant commercial process support systems and research prototypes. Section 9.4 provides a similar analysis for the reuse framework. The whole evaluation is then summarised, showing how the various analyses complement each other and cover the whole scope of contributions.

9.1 Process Modelling

WORKWARE combines ideas from all the process modelling paradigms surveyed in section 3.1. This section compares the language to the most innovative as well as the most representative and widely used notations for work process modelling. The requirements for simple, flexible, user- and domain-oriented models guide this analysis.

9.1.1 Process Modelling and Principles of Sociotechnical Job Design

A computerised information system and its community of users constitute a dynamic sociotechnical system [495, 496]. Models of work processes is a form of job design. It thus makes sense to investigate the principles of sociotechnical job design [495]:

1. *The work system should be the basic unit of analysis, rather than individual jobs*
   We thus need to focus on processes, more than individual steps. WORKWARE gives participants overview of the whole process, and interprets the model holistically. Static and adaptive WMS violate this guideline by emphasising the separation of the process into steps with clearly defined interfaces. This leaves coordination to the automated engine, while users should focus on just their own tasks.
2. *The workgroup, rather than the individual as central unit*
   Emphasising automated coordination, many WMS allocate only one person per task [360]. WORKWARE allows several people filling different roles on each workitem, though the support for teams is limited in the current implementation.
3. *Internal regulation, rather than external control*
   Conventional WMS automate the sequencing of tasks based on externally defined

models. Interactive systems also facilitate coordination through mutual adjustment, by letting users share plans and status information [283, 349]. The EXTERNAL infrastructure includes awareness and communication tools. Such services are also included in some other WMS prototypes [12, 124].

4. *Redundancy of functions, rather than parts*
   This requires that the WMS utilises process models for learning and competence development. The need for allocating personnel to tasks based on competence development plans, not just current competence profiles, is recognised in our work [93]. This enables development of new skills, not just specialisation in a few roles.
5. *Value discretionary rather than prescribed part of work roles*
   As discussed in Chapter 2, prescribed process models often end up biasing official theories over actual practice. Emergent workflow leaves the detailed planning and coordination to those who perform the work, with user-controlled degrees of plan specificity. It is a core principle for interactive models that no syntactic or semantic rule should prevent the users from articulating the most accurate reflection of their practice that they are able or willing to provide. Prescribed process models should be minimum critical specifications only [357]. Such a perspective dominates our work on model harvesting, abstracting practical models into reusable templates. Conversely, adaptive workflow [274] propagates change from generic, prescribed models onto every process occurrence. Hence where conventional WMS enforce strategic change in a top-down process, WORKWARE also supports bottom-up learning from experience.
6. *Individual as complementary to the machine, not as an extension of it*
   Static and adaptive workflow see the automated engine as the force driving the process forward, and users as agents performing predefined steps. Interactive enactment allows the users to complement and control the enactment. Users may proactively override system rules. Rules are also made explicit and can be redefined locally by informed users. In this way the users and the system cooperate in interpreting the model in the situations that arise. From a technical point of view, interactive enactment is needed because one cannot assume that the model is complete. This is a governing principle for the automated reasoning made by the enactment engine: It must always be *open* to the possibility of something else happening that is not part of the model. If such anomalies can be captured by the system, the resulting model will be a richer source for learning than in a system where it is impossible or cumbersome to represent exceptions and situated process innovation. Our emphasis on modelling languages with simple, user-oriented concepts that can be tailored to local contexts further illustrates the focus on users as active, knowledgeable agents.
7. *Variety-increasing both for the organisation and the individual*
   While production workflow aims to build standardised (variety-decreasing) solutions for repetitive processes, this thesis targets knowledge work, where each process is unique.

The rationale behind interactive process modelling thus reflects the job design principles. This shows that the overall approach of this thesis is well aligned with sociological theories about what creates a good working environment. The discussion also pinpoints several important flaws in conventional workflow thinking.

9.1.2 Workflow and Process Modelling Standards

In section 6.4.1, we saw that the standards of the Workflow Management Coalition [530, 532] employ a large number of constructs for modelling items of work, while not providing much increased expressiveness over WORKWARE or APM. We also saw how WORKWARE models can be transformed into the WfMC standard with holistic mapping rules. There are however some areas where constructs in WORKWARE have no translation in WfMC. In particular, user involvement, incompleteness and uncertainty are not handled by the coalition. While they use *AND* and *OR* (actually meaning *XOR*) splits and joins, they cannot represent the uncertainty and need for user involvement if the *unspecified* decision. WORKWARE's decision connectors can be used to represent a number of different WfMC constructs, including split, joins, pre- and post-conditions for workitems, as well as transition conditions for flows.

There are also areas where the WfMC standard has been criticised for lacking precision, e.g. in the semantics of an AND split followed by an OR join [237]. These cases typically deal with scenarios where the whole model, not just individual objects or relationships, must be interpreted. There thus is a need for semantic holism in standardisation efforts as well. Most such problems related to semantic correctness, avoiding deadlocks and ensuring termination, are important for fully automated systems, but handled by user involvement in interactive systems. As pointed out by Jablonski [237], research on semantic correctness [548] only deals with control flow aspects, whereas e.g. resource availability is more important in practice. He therefore advocates more pragmatic approaches to these problems.

OMG and UML Process Modelling Profiles

OMG's Workflow Management Facility targets automation, and is thus mainly relevant for integration with the EXTERNAL infrastructure, where FRAMESOLUTIONS partially follows these standards. Such integration has so far been implemented at a very rudimentary level. Further extension should be feasible, since the standards fit well with our language. Although they are far more detailed and complex, and extend the expressiveness in the area of automation and business object integration, OMG's basic constructs are transformational, and the state transition semantics match ours. The only differences are:

- OMG/WfMC does not differentiate Waiting from Ready tasks.
- OMG/WfMC has one state called aborted and one called terminated. These states involve different rules for subtasks.
- OMG/WfMC allows tasks that have never been active/ongoing to finish.

OMG also has standardised UML [381], where a number of profiles for process modelling has been proposed, e.g. a business modelling profile for UML [381], a Software Process Engineering Metamodel (SPEM) [380], and the Enterprise Distributed Object Computing (EDOC) process profile [379]. Focusing on documenting methodologies in a useable manner, SPEM targets simplicity rather than power of expression, and does not produce executable process definitions. SPEM does not articulate resources and model management, but include a rich vocabulary for modelling units of work. *Lifecycles* are decomposed into *phases*, which may have *iterations*, further decomposed into some levels of *activities*, before *steps* finally denote atomic workitems. Most of these concepts are declared by stereotyping. Through packaging, further classification of workitems into processes and disciplines is possible. Semantic holism with less classification would make this language more flexible and extensible.

BPML - Business Process Modelling Language

BPML [21] defines a formal model for low-level executable processes with web services. Although most BPML control flow structures are also easily modelled in EEML, WORKWARE does not currently support all the constructs. One reason is that low-level automation is left to FRAMESOLUTIONS in EXTERNAL. In particular, BPML transactions and exceptions do not have direct equivalents in EEML. Decision connectors can be used for representing transaction semantics in workitem templates, and the extensibility of the language enables separation of exception handling items from main items, e.g. by adding a property to all exceptional items. BPML supports reuse through *contexts* holding property values. Context decomposition follows the work breakdown structure. Contexts inherit all properties from their surrounding context, similar to namespaces in structured programming. BPML thus exemplifies reuse along decomposition relations. WORKWARE's reuse framework supports this kind of reuse, but is more flexible and powerful.

Standardisation

All of these standards use classification as the primary means for encoding information. This generates larger and more rigid languages, compared to semantic holism. The emphasis on classification is to some extent justified by the need to precisely define the core concepts in a direct manner. By listing the classes of a standard, the required expressiveness is immediately available. Expressiveness achieved through combining classes with properties, constellations and contexts, requires additional description of the different combinations that should be handled. In EXTERNAL we experienced this situation when defining EEML. Although an early agreement defined all the modelling constructs (as object and relationship classes), conflicting implicit assumptions regarding what constellations were allowed, surfaced throughout a long period of time. Flexibility and evolution is also poorly handled by most of these standards because interchange of models is the primary objective. Propagating changes to an ongoing, already transferred, model is outside the scope.

PIF [304] is an exceptional process modelling standard. Aimed at supporting evolution of both models and languages, PIF utilises *partially shared views* for interoperability. With partially shared views, an element not found in the target language is translated into the nearest ancestor (superclass) that has a corresponding class in the target language. Unknown attributes are converted into generic formats, e.g. textual comments. This enables local language extension through specialisation without destroying interoperability. PIF is thus much simpler (measured in number of primitives) than the other standards. In EEML we similarly defined generic extension mechanisms at the metametamodel level. Instance objects may have locally defined properties, and generic classes may thus be specialised intensionally.

9.1.3 Transformational Modelling Languages

WORKWARE's modelling language integrates multiple perspectives. Striving for simplicity and user-orientedness, we take the *transformational* (input-process-output) starting point (as discussed in section 4.1). In addition to the various standards discussed above, a number of other transformational languages have enjoyed widespread adoption in industry and scientific communities. Carlsen [90] compares INCONCERT, TEAMWARE, ACTION and OBLIGATIONS to APM, so they are not included in this analysis.

Petri Net Process Modelling

Petri net is a widely used formal process modelling language. As discussed in Chapters 2 and 3 formality enables automatic activation, syntactic and semantic analysis, but relies on closed system semantics, often resulting in complex models. Van der Aalst [549] provides a formalisation of Event-driven Process Chains by mapping their constructs to Petri nets. WORKWARE's language can be formalised in a similar manner. As van der Aalst shows, OR-connectors (further generalised in this thesis to *unspecified* decisions), result in an exponential growth in model complexity. This example illustrates, that uncertainty can only be represented in formal languages by modelling all the alternative combinations. Consequently, an incomplete model quickly becomes complex and unmanageable. Open system semantics, on the other hand, does not require complete prescription of all alternatives, and may thus articulate uncertainty and vagueness by simple constructs such as WORKWARE's unspecified decision connector.

Other examples of incompleteness, like partially connected workitem collections, are also not allowed in Petri nets. "*Although a finite automaton or Petri net can analyze change from state to state, reachability, and deadlock, it has no mechanisms to analyze the addition of new states nor the alteration of the state structure*" [153]. Consequently, flexibility is generally achieved through other means than process articulation, e.g. manipulations of token configurations [98, 553]. MILANO [6, 124] allows users to model partially connected workitem collections, where the sequencing of items is not completely predefined [7]. Compared to the modelling language of WORKWARE, however, their low-level Petri nets will generate larger models [246], e.g. since state enactment semantics are part of the process model. Standard Petri nets do not support modularisation and abstraction through decomposition either.

Event-driven Process Chains

The function and event constructs of Event-driven Process Chains (EPC) [434] are similar to the workitems and decisions of WORKWARE. However, EPC does not include *states* of functions and events [479], and thus require more detailed modelling. Event conditions can be modelled as decision connectors in WORKWARE. EPC models thus tend to be slightly larger, but they also more directly map the enactment semantics than WORKWARE does.

9.1.4 Modelling Varying Degrees of Specificity

A number of languages seek to articulate both completely predefined and semi-structured, evolving processes. The ability of SEEME [214, 215] to represent *vagueness*, *uncertainty* and *availability of more information* in the models is needed for creative processes of articulation, negotiation and social reality construction. SEEME's solution, to attach secondary symbols denoting vagueness etc. to primary model elements, enables integration with any graphical language, including that of WORKWARE. This can be an optional feature; it need not complicate the models. In WORKWARE, incompleteness, vagueness and uncertainty are assumed characteristics of all models. This means that we currently cannot differentiate between fixed and changing parts of the model, they are all interpreted the same way. Linking enactment policies to vagueness and uncertainty properties could make the enactment of predefined procedures more straightforward. Further case studies are needed to investigate to what extent such properties can and should be explicated. Experience shows e.g. that making it explicit that some information is not articulated, increases interest in that which is hidden, contrary to the objectives of the modellers [216]. It is not evident whether the default interpretation of a workflow fragment should be that it is completely or partially specified. Often this property should be reused, e.g. from the surrounding process.

Herrmann [214] present concepts for *evolving workflows* that utilise SEEME for "*user-driven coordination of incompletely-prescribed, non-anticipatable business processes*", stating that such support can "*only be successful if those employees who actually perform the tasks of the processes, regularly make contributions*". This concept closely resembles interactive and emergent workflow, however it has not resulted in detailed designs or implementations. On the other hand, numerous case studies that utilise the SEEME modelling environment support claims made in this thesis [215-217]. For instance, it was discovered that people are able to articulate their own work in process models, provided initial training and some starting point models. Sequencing of activities, a cornerstone in conventional workflow management, was also studied. It was found that process experts often artificially construct sequences by "*projecting the sequence of an interview onto real work situations or by assuming logical dependencies which do not correspond with reality*" [214]. Their research thus complements the work presented here with independent case studies, leading to similar conclusions and requirements.

Constraint-Based Process Modelling

As outlined in section 3.1.4, constraint based PMLs have been proposed to increase the flexibility of process articulation. Constraint specifications define a space of allowed behaviour, which collapses onto one chosen solution only when the time for action arrives. In GPSG [183], constraints are used for expressing "soft dependencies" among tasks. GPSG aims at supporting the whole spectrum from unstructured work facilitated by groupware, via semi-structured to fully structured processes. Open process definitions allow rules to be added and removed. By defining declarative rather than operational rules, GPSG is able to express the organisational context within which a work process is situated. A variety of rules handle task interdependencies, e.g. whether a flow from A to B means that B starts when A finishes, that B should not finish before A, that B starts after A starts etc. In this way, a flow from A to B may influence both A and B. Tang and Hwang [484] extends the set of task dependencies to include transaction support (abort and commit). However, declarative constraints offer little support for process participants to jointly articulate *how* they are going to perform their work, their project plan. Consequently, GPSG also supports decomposition, conditional and parallel branching and operational triggering of activities. GPSG models are used in a *separate design step* that provides input to automatic generation of an executable plan by constraint solvers. However, users may also alter the executable models.

Compared to GPSG WORKWARE offers *simpler, visual articulation* support and user involvement in enactment. We are aware of no other operational PML that provides the same level of flexibility. Often users find it easier to articulate what they intend to do, rather than all the contextual rationale behind it. This is evident in the current practice of project planning. Constraint-based languages are most suited for modelling the external influences on the project. These approaches thus emphasise process modelling by outsiders. Since the cost-benefit trade-offs involved in process articulation may result in under-constrained models, and the conflicting interests of stakeholders can result in over-constrained models, constraint-based approaches should be complemented with negotiation tools and *interactive* process support. GPSG could function as a model interactor in WORKWARE, reminding users about constraints from the organisational context (e.g. approaching deadlines), and supporting them in resolving these constraints.

System States and User States

FREEFLOW [142] also applies constraints. Its task enactment metamodel allows some user-control of enactment through the separation of system and user state. The system state (*disabled*, *enabled* or *pending*) is the result of causal dependencies among tasks, while the user state (*inactive*, *active*, *done*) defines temporal states caused by user actions. In WORKWARE, system states are represented by the input and output connectors of workitems. User states are reflected in the states of the workitem. WORKWARE supports the state set of FREEFLOW, adding abnormal termination. This allows us to handle early start of the work (opportunistic involvement), where the user state becomes active although the system state is still disabled.

Goal-Oriented Process Modelling

Goal-oriented, declarative process models [231, 303, 543] assign goals and constraints to agents, but leave them to work out for themselves the details of how to reach these goals. In WORKWARE, decisions represent conditions, and although the current prototype has not implemented a full rule language, section 4.6 shows how rules can be modelled. WORKWARE's operational syntax also defines what should happen when rules are violated. Such rules are more transparent, since the users see the consequences of the rules as well as the conditions. Transparency is important in an interactive system. ALLIANCE [11] reflects the vagueness of explicit process models by capturing all knowledge about the current state as *fuzzy* statements with truth probabilities between 0 and 1. Fuzzy logic opens up a richer set of alternatives to the agents that interpret and activate the model. WORKWARE currently lacks this feature, so uncertainties must be handled manually.

9.1.5 Language Action and Conversational Process Models

The COORDINATOR [538] and its successor, the ACTION workflow system [339], focuses on articulating and supporting communication and negotiation among the actors of a work process (cf. section 3.1.2). Their work breakdown structure influenced WORKWARE templates and reuse policies to include customer and performer roles. Carlsen [90] show how such conversations can be modelled in APM with pluggable actions. WORKWARE allows us to simplify the modelling of conversation steps by representing them as decisions, and typical conversation patterns [339] have been operationalised by WORKWARE's standard enactment semantics. By modelling decision-making responsibility, the roles of customer and performer become more clear, similar to role-oriented modelling. Interactive enactment with holistic enactment rules also avoids some of the criticism towards the language action approach (cf. section 3.1.2). WORKWARE adopts information sharing, as opposed to a message-oriented approach, and does not control or constrain the communication between users.

9.1.6 Decision Oriented Process Modelling

Explicit representation of decisions is a key enabler for the interactive activation mechanisms described in this thesis. Decision oriented process modelling have been proposed in some research prototypes. In section 3.1.3 we saw how issues, positions and arguments capturing the rationale behind decisions could be captured in decisions-making loops. Such approaches could complement WORKWARE, e.g. to capture the argumentation and reasoning behind key decisions, for instance in selecting which template to reuse. By modelling decision-making loops as a decomposition of WORKWARE decision connectors, the context of issue, alternatives (outputs) and related resources would be available as a starting point for the argumentation articulation. RAPCEE [401] models form a generic work breakdown structure with AND (executive, automatic) and XOR (manual choice) contexts, similar to the reuse decisions presented in section 6.2.1. Although RAPCEE supports both manual and automated enactment, WORKWARE is more flexible in allowing situated automation boundaries, letting users override executive plans and partially automate choices. Unspecified decisions allow partial articulation of uncertain process fragments.

9.1.7 Other Process Modelling Languages

From role-oriented PMLs WORKWARE has adopted the technique of using graphical containment to denote responsibility. The main difference between e.g. RAD [391] and WORKWARE is the primary structure for organising models: roles or work breakdown. WORKWARE models could however be visualised in a role-oriented manner. For some processes, e.g. communication, negotiation and administrative routines, the work breakdown and role structures are well aligned. For more flexible and knowledge intensive work, we find that work structures are more basic and stable than roles.

Object oriented process modelling (cf. section 3.1.7), utilise a large number of different diagrams, reflecting the need to provide suitable dialects for different modelling perspectives. EXTERNAL facilitates this through different views and interfaces to one *integrated* model, thus substantially simplifying the conceptual framework. This solution obeys principles of semantic holism and perspective integration.

9.2 Interactive Modelling Languages

In this section we turn from process modelling to the application of interactive modelling principles to software and enterprise modelling. The aim is to show that the proposed principles enable simpler and more flexible articulation in these domains as well.

9.2.1 Semantic Holism for Enterprise Modelling

The METIS Generic Enterprise Modelling (GEM) template [347] reflects 10 years of experience in utilising an open enterprise modelling tool [71, 309]. METIS allows user organisations to add their own constructs to the generic ones in the template, and the elements of GEM have been harvested from a number of different industries. The openness of the METIS tool also facilitates in-depth studies. GEM consists of 19 modelling domains, reflecting different aspects of an enterprise.

The domains contain a total of 73 object types. Here we investigate to what extent semantic holism can help us reduce this number. 30 of the object types can be decomposed. Mostly, GEM utilises semantic holism to limit classification along decomposition relations, so there are no separate types for composite and atomic objects. There are however 4 exceptions to this in the product, competence and information domains. Objects and relationships in METIS models are *instances*. This means that separate modelling constructs for classes are not included, cutting the number of constructs in half. There are however 3 exceptions to this rule as well. So far the GEM template still seems to utilise semantic holism quite well.

When we examine the set of constructs more closely, however, a different picture emerges. In process modelling, three different schemes are available, the flow logic language based on IDEF0, a workflow language for operational models, and the IPM domain which integrates concepts from the two others. By integrating these dialects, we can remove at least 17 types.

For the 4 document types, properties could replace classification as encoding scheme, making the language more flexible. In other cases the separation between two types can be derived from the context. Online and offline documents are stored differently, catalogue parts are external product items, machines and tools are just systems in a different context, organisational positions reflect similar relations as process roles etc. Goals/intensions and requirements could also be combined, since their difference is expressed by relationships to other objects. There are also generalisations that exist for the purpose of articulating common roles of a number of different types, e.g. *Work unit*. While these generalised types may be useful for expressing uncertainty, this can also be achieved through other means. Finally, reflection would allow concept, property, information element, and label to be represented by metametamodel constructs.

This analysis shows that semantic holism allows us to remove at least 36 of the 73 types in GEM, without loosing expressiveness. All the principles from Chapter 6 were applied. There may of course be pragmatic reasons for including all of these different types in GEM; e.g. that it should be simple for users to understand what they can express with the language. Templates with different default properties, could however achieve this in a more flexible manner.

9.2.2 Simplifying UML with Semantic Holism

UML is a standard for visualising, specifying, constructing, and documenting software systems and applications [381]. An analysis of UML is a suitable means for evaluating the generality of the modelling approaches proposed in this thesis. As we discussed in Chapter 3, UML is also applied for enterprise and process modelling. The current version of UML suffers from a number of weaknesses that are especially important for interactive modelling, including:

- *Size and complexity*. The UML core (version 1.4) defines more than 200 different primitives [281] and 9 diagram types [137, 342].
- *Rigidity*. UML is class-based, making local instance modifications and conceptual evolution cumbersome. While an object may be instance of more than one class, all behavioural and structural features must be defined at the class level.
- *System-oriented*. UML and its predecessors were primarily developed to make it easy for programmers to document their code and designs. Consequently most primitives mimics those of object-oriented programming [137].

The primary aim of this section is to show that semantic holism can be utilised to fix some of these problems.

Simplicity

The context where a model element is placed should influence its semantics. In UML [381] a number of contexts are defined for structural modelling, including *class diagrams*, *instance diagrams*, *role and collaboration diagrams*, *stereotypes*, *templates*, and *implementation diagrams*. In these contexts we find alternative ways of denoting objects (Class, Instance, ClassifierRole, Stereotype, Component, ComponentInstance), relationships (Association, Link, AssociationRole, Binding, Deployment), relationship endpoints (AssociationEnd, LinkEnd, AssociationEndRole), properties (Attribute, AttributeLink, TagDefinition, TemplateParameter) and values (InitialValue, Value, TaggedValue, TemplateArgument).

Here the lack of holistic and contextual semantics requires a large set of primitives and duplicate encoding. It is however interesting to note that the constructs for extensibility, evolution and incomplete models, utilise semantic holism to a greater extent than the rest of UML. For instance, Templates are not defined atomically as first class constructs, rather any element that includes TemplateParameters is a template. Stereotypes and TemplateParameters can both be attached to any ModelElement, e.g. to classes, associations, roles and instances. Another proposal for extensibility in UML lets different packages describe separate perspectives on the same elements (e.g. with different attributes) [144]. In order to define domain specific profiles, modellers extend standard packages as whole, not every individual element. This holistic, contextual technique produces simpler models than conventional subclassing.

The 3C (*Clear, Clean, Concise*) proposal for UML version 2 [170, 383], takes an even more holistic approach. With 15 primitives, 3C support the full richness of UML, with improved extensibility. The proposal views model elements as representations of individual, observable phenomena, not as specifications of general system components. Consequently, the basic model elements *object*, *association*, and *action* are defined as instances. A generic *type* definition applies to all three basic elements, and classification is not inherent. *Attribute* is defined as a special kind of *association*, and *class* as a special kind of *template*. 3C also separates inheritance (of implementation *classes*) from specialisation (of classification *types*). Although 3C aims at defining a precise, formal notation, their perspective is far better aligned with interactive models than the existing standard. Their work shows that semantic holism also has a role to play in more conventional modelling frameworks.

Comprehensibility

In UML version 1.5, the semantics of the relationships between the class and the instance context are dispersed among the definitions of *Instance*, *Link*, *LinkEnd* etc. Such inter-context relationships should be defined in one place, holistically for the whole context, rather than separately for each construct. A more principled definition would enhance readability of the specification and make it easier for novices to grasp the basics. The separation also creates a need for a lot of specialised well-formedness rules to ensure consistency [342], where general principles could suffice.

Flexibility

Inherent classification, where the class defines all features that instances may possess, makes typical lifecycle development more difficult. During initial modelling, it may not be easy to determine whether something should be represented as a class, a template, a stereotype, or a role. Migration between these metaclasses should thus be supported. Currently, the semantic holism evident in the contextual and dynamic definition of *Template* allows migration between class and template, but the other transitions require more work. With increased holism, automatic transformation of model fragments between contexts, e.g. a class diagram to a role diagram, could be achieved by simply moving it to another context. Similarly, a scenario model (instance context) could be automatically generalised into a class diagram.

Expressiveness

By removing some of the differences between e.g. instance and class, new capabilities are automatically included, without making the language more complex. For instance, features (attributes and methods) could also be declared locally for individual instances.

Summary

This brief investigation of structural modelling in UML shows that semantic atomicism makes the language complex and poorly structured. As with many standardisation efforts, direct expressiveness is highlighted, while flexibility and simplicity, seems not to have influenced the UML design to the same extent.

9.2.3 Interactive Conceptual Modelling

The modelling framework proposed in this thesis combines a number of techniques from previous research, including instances with dynamic property sets [385, 393, 396], semantic holism [263], and interaction [524]. These frameworks are compared to the WORKWARE metametamodel in section 9.4. As the discussion above shows, none of these techniques have been in widespread use in process modelling. Our original perspective of incomplete, evolving, operational models has also extended these theoretical frameworks. Instances have been combined with incremental classification to meet practical requirements for customisation and reuse, while property modelling has been coupled to semantic holism with contextually derived properties. Identifying practical techniques for semantic holism is a further contribution of this thesis.

9.2.4 Summary

Sections 9.1 and 9.2 have shown that WORKWARE provides extended flexibility in a simple language compared to other process support environments. The approach goes beyond current systems in facilitating partial, and evolving articulation of work processes. The language includes the most interactive features from a variety of process modelling paradigms. It shows that transformations, language action, roles, rules, and objects can be integrated without making the language overly complex. This integration is enabled by interactive and holistic semantics. The foundation of tried and tested modelling techniques is an argument in favour of the feasibility of the approach.

We have also shown that instance modelling with flexible assignment of properties is relatively new in process support. This technique solves some of the requirements that previous approaches could not meet, most notably simplicity, local modification, and conceptual evolution. Languages that support incomplete workflow specifications are rare, and no operational language was found that integrates the whole spectrum from unspecified to fully predefined. The implications of allowing incomplete models are substantial, removing problems of over-serialisation, disintegration, external control, and lack of discretion reported in the literature. The weaknesses of the WORKWARE language include a sometimes too heavy reliance on user participation, and that the general language may be too complex in early phases of modelling. Reuse and alternative visual representations of models are needed to remedy these problems.

By exploring the application of holistic techniques in UML and enterprise modelling, general applicability is assessed. This approach reflects a fundamental trend in conceptual modelling to focus more on the pragmatic utilisation of models and less on their linguistic and referential aspects [102, 221, 224]. Most work that compare process modelling languages look at expressiveness [115, 119, 195, 265, 305, 479]. Other aspects like flexibility, simplicity and user-orientation are seldom considered. Large, complex languages benefit from such comparisons, where it is generally considered correct to have separate classes for each concept that one would like to articulate. A more general approach for assessing process support systems were proposed by Carlsen et al. [96]. They extend the model quality framework discussed in Chapter 2, including e.g. pragmatic and social aspects in addition to expressiveness. In this thesis our focus is the integrated articulation and activation of interactive models, thus the requirements are directly in conflict with the conventional assessment method. Instead, a case-by-case comparison between WORKWARE and other languages have been performed. A similar shift from formal studies of expressiveness to pragmatic engineering simplification was experienced in programming language research around 1970 [521].

9.3 Interactive Activation

In the workflow literature, process model activation mainly deals with fully automated enactment. This thesis proposes interactive enactment of incomplete, evolving models as an alternative perspective. In addition to increased flexibility, WORKWARE aimed at integrating a larger set of model driven functionality.

9.3.1 Interactive Workflow Enactment

Bernstein [45] combines coordination theory and soft constraints, enabling a flexible distribution of control between the system and its users, handle varying degrees of model specificity (cf. section 3.3.4). While our focus has been end user articulation and model interactors, Bernstein relies on model reuse and constraint-based advice. While targeting many of the same problems, his work assumes that end users are capable of selecting among alternative predefined processes, but cannot be expected to model their own work. Bernstein's architecture complements WORKWARE with a different set of model interactors. Our work goes further in implementation and use, and we have designed one modelling language for all degrees of specificity.

Computational coordination mechanisms in ARIADNE [135] can be incompletely defined. When some information is lacking, users are asked to provide it at runtime, similar to WORKWARE's fallback mechanism. However, Divitini and Simone claims that significant user contributions at runtime "*is in general not reasonable*" [135]. Consequently, the modelling in their case studies are performed by the researchers themselves, and the language makes few concessions towards end user participation. There is no construct similar to WORKWARE's decision, which facilitates and focuses fallback questions.

On the other hand, some designers of process support systems [214, 340], stress the potential of visual models to enable user involvement. ECHOES [340] utilise instance modelling to support local modifications. However, users participate in model articulation, and not in unfolding activation of the models. On the other hand, ECHOES offer better means for process automation, e.g. service modelling, user interface content and layout definition [340]. The EXTERNAL infrastructure needs extensions in this direction, filling the gap between WORKWARE's flexibility and FRAMESOLUTIONS hard-coded automation.

Antunes et al. [16] argue that the separation between workflow support for formal procedures and groupware support for informal processes is artificial. Their ORCHESTRA engine identifies situations where formalised solutions do not exist. When such an exception occurs, the ORCHESTRA MATCHER selects the most appropriate group interaction technique, tools and actors to solve the problem. The system thus includes interaction in its model activation portfolio. Its group interaction support (e.g. brainstorming, cognitive maps, and deal-making negotiation) could be a useful interactor in the EXTERNAL infrastructure. However, ORCHESTRA relies on models built by external experts, and does not support articulation of ad-hoc tasks.

Linked Abstraction Workflows (LAW) [164] allow explicit modelling of process participants' authority to change the model during enactment. Our use of the graphical location of connectors to denote decision-making authority seems simplistic by comparison to LAW's sophisticated mechanisms. But simplicity is also the strength of our approach. We achieve rudimentary support without complicating the modelling language. More sophisticated support is given by the access control interactor. Case studies are needed to investigate what level of detail should be put into explicit representations of authority, as this can be a highly political issue.

OBLIGATIONS [59, 58] supports ad-hoc evolving webs of interdependencies among tasks. Any state transition in one task can be related to any transition in another task. New obligations, where one person or group commits to performing work requested by another, can be dynamically added at any time, but once they have been assigned to a performer, they can no longer be modified. While it contains a richer vocabulary for coordination than WORKWARE, the system is thus not quite as flexible. Opposite to this and most workflow research, we started with ad-hoc processes and added support for routine procedures later. This lead to a different design, e.g. with a simpler modelling language and more manual control.

INCONCERT [1, 429, 430] stands out among commercial workflow management systems when it comes to supporting ad-hoc processes. INCONCERT models process instances, enabling *process design by discovery*, starting the work with a rudimentary template and elaborating it as the work progresses. The modelling language may be extended with user-defined subclasses and attributes. INCONCERT also provides generic and customised task user interfaces similar to WORKWARE's worktops. When exceptions occur in the execution of automated tasks, reactive user involvement is applied. However, proactive decision making appears not to be facilitated. In 1994, Abbot and Sarin outlined a number of challenges for the next generation workflow systems, based on early experiences with INCONCERT [1]. WORKWARE brings contributions to integrating procedural and non-procedural work, utilising general mechanisms in customised policies, access control integrated with process models, evolutionary process development, and a library of reusable templates based on experience. The EXTERNAL infrastructure also supports external activities and meetings. Change management, configuration and version control is the only challenge not yet met by our work.

HIERASTATES [486] is a formalism for model-driven, state-dependent object behaviour, which uses STATECHARTS [209] actively at runtime, e.g. to present alternative actions to the user. The language is refined with non-atomic transitions that interact with users during their activation, enabling users to stop the transition or select among alternative targets. The TACTS workflow prototype is built on top of HIERASTATES in order to support a mix of structured and unstructured activities, taking a clearly interactive perspective [487]. TACTS' support range from no explicit process, via the process model as a guide, to partially and fully automated processes. In order to make STATECHARTS interactive HIERASTATES adds constructs for exception handling and allows partially structured models. Processes are modelled at the instance level. WORKWARE goes beyond TACTS in providing a full-scale work support system, not just a prototype dealing exclusively with the process dimension (tasks and flows). Consequently, we have developed richer modelling languages and dealt more with the problems of end user articulation. The enactment semantics of WORKWARE involves users to make decisions directly without having to change the model, while TACTS just relies on user involvement in modelling. WORKWARE's enactment engine does not make active use of the state transition model, which is hard-coded into the system. A more open, rule-based enactment engine is thus a primary objective for our further development efforts. HIERASTATES also provide flexible reuse mechanisms, which are discussed in section 9.4.

9.3.2 Adaptive Workflow Enactment

A few adaptive workflow systems have also applied interactive techniques. In MILANO [7] exceptions are handled by user-controlled jumps, policies control users access to make jumps, and negotiation support is integrated in the exception handling process. Differences between adaptive and interactive workflow are however evident in how models are interpreted. MILANO requires that all allowed paths be computed from the workflow model. WORKWARE recognises that the set of paths cannot be predetermined in an open system. Instead the chosen path unfolds in step-by-step situated activation. Users may override the current model, by explicitly making a decision, activating a flow, or starting a work item.

Reflection in Adaptive Workflow Enactment

ENDEAVORS [261] and NETS-IN-NETS [554] are examples of reflective workflow systems, which enable dynamic model interpretation. They allow the engine to be configured at runtime, selecting which set of enactment rules should be enforced. WORKWARE complements this with *implicit recongifuration*. It integrates several enactment rules into the basic enactment policy, and allow different rules to be applied in different situations. Situation patterns are recognised from a holistic view of the model. Hence, users need not explicitly switch to another execution policy. They just change the model or its elements' states to reflect the current situation. For instance, a user clicking on the "Start" button of a not-yet-ready workitem causes opportunistic involvement, and a re-interpretation of the violated flows. Experience indicates that this domain-oriented approach is simpler to use than explicitly changing execution strategies [1].

RECONFIGURABLE NETS [26] is a class of Petri nets that can dynamically modify its own behaviour by rewriting some of its components. For such nets, boundedness and soundness can be determined, i.e. it is a primary objective to guarantee that the model can be enacted in a closed system. However, since the flow relations in the model depend on the whole marking (set of tokens in the model), a form of dynamic semantic holism is achieved. This enables e.g. routing decisions of process instances to be made depending on the amount of work currently queuing in different parts of the process. The ROK framework [148] utilises reflection through a metaobject protocol [269] to support *programmers* in ongoingly adapting workflow definitions to changing business needs. In a design for extending the flexibility of the BAAN workflow engine [341], *shoot tip activities* trigger meta-activities for defining the next steps of the process. The shoot tips are growing points for the process model, allowing ad-hoc modelling at predefined places. In the EXTERNAL project we have similarly modelled the meta-process of project planning, but here changes are allowed at any place in the model. We allow any task to become a shoot tip, by supporting concurrent modelling and execution. Similarly, planning tasks are modelled just like ordinary tasks. BAAN also jumps to the meta-level when exceptions arise, providing transaction support, which is not implemented in the EXTERNAL infrastructure.

Exception Handling in Adaptive WMS

Exception handling (cf. section 3.3.2) is the area where adaptive WMSs most commonly utilise interaction with end users [66]. Emergent processes cause frequent exceptions in such systems. Cugola [117] argues that deviations and inconsistencies should be tolerated and supported in process models, but his systems only handle temporal deviations from an otherwise complete model. WORKWARE treats exceptions as the rule of the game, but lacks sophisticated mechanisms for controlling the handling of rule violations. An exception handling interactor would be a useful extension to the EXTERNAL infrastructure. The work of Borgida and Murata [66] is relevant in this respect. They define *dynamic classes* for each of the states that a task may have, and define exception-handling rules based on these primitives. A similar dynamic classification scheme (intensions) is already available in WORKWARE. The state transition model of [66] also matches ours, so integration should be straightforward, at least on the conceptual level. WORKWARE's framework also handles reuse of exception handling rules and procedures.

End User Customisation of Workflow Interfaces

Dangelmaier et al. [120] highlights the importance of accommodating both personal preferences and domain nuances in workflow systems. They develop a multi-level preference model, where customised instances of system components are selected for different workflow states. WORKWARE's customisation features utilise more dimensions of the current state, and its reuse framework also is more general. On the other hand, Dangelmaier et al. have put more work into user interfaces for end user tailoring. WORKWARE just provides raw data object editing facilities for policies, so a more specialised, process-driven interface is needed.

9.3.3 Extending Activation beyond Enactment

Other tools provide more complete collaboration support than the WORKWARE prototype. For instance ORCHESTRA [16], OBLIGATIONS [59, 58], XCHIPS [233] and MILANO [6] provide a richer integration with conversation tools and shared workspaces. The strength of WORKWARE lies in its alignment with email and Internet technology (simplicity), and the *service* concept, which allows uniform configuration of and access to internal and external functionality. The integration with XCHIPS in the EXTERNAL infrastructure has also provided more sophisticated communication support in the context of workitems and processes.

Integration of workflow and awareness notification is proposed by a number of researchers [415, 416, 428]. Our work goes further in elaborating the detailed usage scenarios where such a component plays a role, and in articulating the interplay of awareness and enactment. WORKWARE's awareness customisation approach is inspired by TVIEWS [518], but utilises holistic event propagation through complex structures to a greater extent. INCONCERT, INTERMEZZO [149] and some specialised approaches [49, 68] have combined workflow with access control, although not with the same flexibility and reusability as our implementation. Project management has also been prototyped in LAW [164], and INCONCERT is integrated with MS PROJECT. Time, resource and project management have been designed, but not yet implemented in WORKWARE.

9.3.4 Towards Interactive Workflow Architectures

There are many layered workflow architectures, and as the previous section shows, WORKWARE is not the first system to extend active support beyond enactment. The novelty of the WORKWARE architecture lies in the *interactor* concept. It captures the integration of process articulation and activation. Another key feature is that an interactor never assumes to have complete control of the model. Instead different interactors complement each other in a context-dependent manner. The architecture separates functionality from the model objects, following the *strategy* design pattern [173]. The assignment of functional features to model objects is determined by policies. This flexible binding enables personalisation and contextualisation. ENDEAVORS [62] similarly decouples object data from the *handlers* that control behaviour. Less flexible environments, e.g. FRAMESOLUTIONS and E^3 [238, 241], directly map model elements to software components.

Software agents [372] also distribute functional support among semi-autonomous components. An agent has some degree of autonomy and adaptability, and it interacts with the environment [458]. A number of research prototypes have applied agents for flexible process support [230, 282, 513]. In the COORDINATION LANGUAGE FACILITY (CLF) multiple agents interact to facilitate distributed enactment in virtual enterprises [65, 192]. ALLIANCE [11] provides similar malleability, distribution and decentralisation for software-intensive processes, utilising fuzzy logic to reason about uncertain and incomplete models. The ALLIANCE architecture includes agents for user interaction, tool integration, enactment, timing, monitoring, process change and decision making. Agents are dynamically assigned to tasks in the process model, but activation is separated from articulation, so these agents are not designed as model interactors.

The current WORKWARE implementation has a number of limitations. It lacks rich, model-driven integration with external systems based on standards for object management and web services. The utilisation of such interfaces should not be a major problem, given the conceptual alignment between the standards and our modelling language. Scalability is another key problem. The flexibility of an instance-based metametamodel may cause performance problems. In the EXTERNAL case studies, we experienced poor performance when large models were stored in files on a web repository. File storage fits the access patterns of modelling tools, but work performance demands more fine-grained access. While it is straightforward to define a relational database schema for WORKWARE's metametamodel, experimentation is needed to verify how many model objects a typical server configuration can support.

Another shortcoming of the current EXTERNAL infrastructure, is the poor support for distribution of functionality across multiple servers. This is especially important for inter-organisational cooperation, where ownership to sub-processes is distributed among the partners. A number of commercial systems and research prototypes for distributed workflow enactment have been developed [147, 175, 177, 261, 318, 408]. Our work has concentrated on the logical architecture of the system, not the physical level. However, it is evident that the two are interdependent, in that local autonomy may demand both physical and logical distribution. Alignment of our infrastructure with the research in this area is thus a major challenge for further work.

9.3.5 Other Activation Approaches

Workflow management is not the only area investigating model- or data-driven flexibility. This section gives a brief overview of flexible groupware and information systems.

Reflection

In knowledge-based systems, the operational logic is stored as data rather than programmed in software. Reflective systems expose representations of their own logic to their users, and allow modification of this logic. Reflection has been applied at the implementation level of groupware toolkits [138], increasing application programmers' control. In our approach, reflection is applied at the modelling level. Through metaobject and metamodel interfaces, users can add and remove properties on instances and classes, update classes in the modelling language etc. Interactive models thus bring flexibility to users like metaobject protocols bring flexibility to programmers [269]. Reflection at the implementation level would however be a useful extension in increasing the level of automation in our infrastructure.

Customisation

A number of tailorable information systems apply techniques described in this thesis [259]. In AMULET and GARNET [358] instances are the primary objects, and Teege [488] shows the utility of *feature composition* for customisation, configuration, and extension. Feature composition allows properties and behaviour features to be added dynamically to objects. But while these systems apply software-oriented terms in their customisation, interactive modelling utilises a domain-oriented language.

DIVILAB [399] defines a framework for domain-specific visual languages with operational semantics. Their hypertext metametamodel consists of objects and relationships. For each object type, users can define model, view, and controller aspects. HYPEROBJECT SUBSTRATE (HOS) [453, 454] is another flexible hypertext system. It facilitates *incremental formalisation* by end users. Starting with an unstructured model, users may add visual and textual features as they see fit. As the work progresses, text can be converted into lists, then further into objects with attributes, relationships and even inheritance. HOS uses prototype inheritance to facilitate this. EXTERNAL does not support incremental formalisation in modelling with the same level of sophistication, but WORKWARE's metametamodel has the same capabilities as HOS. These techniques have been thoroughly validated [451, 452, 454].

OVAL [330] is another framework for cooperative applications. Its basic constructs are objects, views, agents and links. Usage experience with OVAL shows that nearly every time a user wants to perform a schema operation such as adding an attribute or setting default values, he is already in the context of a specific instance. This caused OVAL to include reflection operations on the instances, while remaining to separate class schemas from instance data. WORKWARE supports objects with links as attributes, and the user interface policies control dynamic composition of views. Its pure instance-based metametamodel simplifies customisation compared to OVAL.

Policies, Media and Mechanisms

In groupware design, *mechanisms* that directly support existing work practices in many cases become too inflexible. Instead *media* that can be tailored to suit each participant's needs, making the cooperation policies and protocols adaptable at runtime, have been proposed as an alternative metaphor [1, 41, 149]. Following this scheme, each model interactor in WORKWARE is controlled by a set of policies. Policies are themselves model objects and their scopes are determined by reuse policies. Conventionally, policies are more system oriented than primary model elements. This language problem has been recognised as a general limitation for deep tailorability (below the user interface) [41]. While a number of organisational and social issues interplay in overcoming the barriers to end user customisation [493], designs that integrate customisation with work performance are found to be important [41].

Dynamic Object Models

Dynamically interpreted and active object models have been utilised for adaptive workflow systems [280, 333]. Experience indicate that such designs work better with intellectual processes than in manufacturing [333]. Like WORKWARE, these dynamic object models (DOM) use the property, strategy, metadata and visual builder design patterns [173]. However, by allocating strategies to types rather than instances, DOM becomes more complex and rigid. WORKWARE's reuse and access control mechanisms can be configured to mimic DOM, but support more flexible schemes as well.

9.3.6 Summary and Assessment

WORKWARE extends conventional techniques for customisation, integration, and extension with model-driven *contextualisation*. A visual model where domain knowledge is articulated influences the operation of the system. Powerful reuse mechanisms and semantic holism are further contributions of this work to the design of evolving and tailorable information systems. A number of the systems above provide better support for automation than the current WORKWARE prototype. However, the interactive enactment concept improves the support for emergent, ad-hoc processes. Such support is seldom prioritised in existing designs, although it is often mentioned as a requirement. The complexity and rigidity of modelling infrastructures effectively hinders end-user participation in articulation and activation.

While research projects in adaptive WMS have proposed human intervention, e.g. in the handling of exceptions, this is commonly a last resort, when all else fails. Utilising user involvement to simplify the languages and mechanisms of the WMS, and integrating support for manual interpretation, is a new research topic for the workflow community. Widening the scope of model-driven support, is another reassessment that this thesis contributes to.

9.4 Reuse and Process Knowledge Management

This section explores how WORKWARE generalises current model reuse mechanisms, and points out the increased applicability that this entails. Most PKM methods focus on documenting experience (articulation). Some reuse and activation methods exist, but most lack flexibility. WORKWARE's integrated PKM solution is need-driven and anchored in practice. By emphasising both accurate capture of processes and situated search, selection, and appropriation of templates into each project, a solution better matching the nature of knowledge intensive work emerges.

9.4.1 Metametamodels for Workflow and Other Modelling Domains

Features and qualities of different meta-languages for process support have surfaced during the preceding discussions. Here our focus is harvesting and reuse. Few workflow researchers emphasise metametamodels when publishing their designs. This is in part because many use conventional languages like Petri nets or UML, where the semantics are already defined. The rigidity, system-orientation and complexity of these techniques motivate a fresh look also at the underlying metametamodels.

ENDEAVORS [62] is one of the most adaptive workflow systems on the market. Its metametamodel includes multiple inheritance, dynamic definition of properties and behaviour, enabling *"better support for reuse than the traditional class-instance model"* [62]. Both ENDEAVORS and WORKWARE separate object data from behaviour, facilitating reuse of one aspect at a time [222]. Harvesting is accomplished by *category promotion*, where an instance model is elevated to the process specification level as a new template. However, attributes and methods can not be defined at the instance level in ENDEAVORS [542]. WORKWARE thus has a simpler metametamodel, allows more local modifications and more general inheritance. Both systems use change propagation rather than delegation for dynamic change.

HIERASTATES [486] defines all behaviour at the class level, but instances are dynamically added and removed from classes. Each property is defined as a class, so multiple inheritance is needed to combine the features of an object. Since all properties are classes, any modelled aspect can be utilised for inheritance, similar to WORKWARE. The separation of classes from instances makes HIERASTATES less comprehensible [453]. The lack of mechanisms for lifetime sharing (propagation) from classes to instances, is another limitation.

In method engineering [74, 371], metamodelling is a means for adapting general methodologies to particular projects. METAEDIT [322, 492] is a commercial CASE (computer aided software engineering) shell that has performed successfully in adapting IS modelling to a number of domains [346, 345]. METAEDIT uses OPRR (*Object*, *Prop-*

erty, *Role*, *Relationship*) as its metametamodel. OPRR is conceptually on a more abstract level than the instance-oriented metametamodel of WORKWARE. It is suitable for defining modelling languages as a set of classes, but offer limited support for instance-level modifications and extensions.

Instance Modelling

In conceptual modelling, the work of Parsons and Wand [393, 396] on instance models and classification theory has been an important source of ideas for this thesis. However, their work on criteria for good classifications, is based on the assumption that a complete object population can be specified [393, 394]. For complete models, criteria for theoretically sound classification structures are defined. In an open and evolving model we must also allow under-specified classes that represent properties known to the users but not yet articulated in the model [383]. As discussed in section 5.2, the need for incremental, evolving classification structures motivates intensional and extensional classes in WORKWARE.

OPJ (Operational objects) [5] is another instance-based modelling approach. The models are operational, and have been used for workflow modelling [79]. OPJ targets software development for runtime flexibility rather than interactive operation. Instance models are used at the system level, based on class level domain models. Reuse of parameterised building blocks is facilitated at both levels. WORKWARE facilitates local modifications at the domain level as well.

Semantic Holism

Kangassalo [263] introduced semantic holism to conceptual modelling in order to capture problems with global understanding, concept creation processes, subjective points of view, and conceptually closed information systems. His CONCEPT D language [262] defines new concepts incrementally from primitives and existing concepts by *intensional containment* relations. Such relations express specialisation, composition, properties and other associations. Concepts are thus not classified into e.g. objects, properties and relationships, making the metametamodel extremely simple. Niinimäki [369] distinguishes between extensional (members), intensional (properties), and hybrid modelling languages. CONCEPT D is purely intensional, while WORKWARE is a hybrid. Niinimäki concludes that while intensional languages map well to the semantics of databases, hybrid languages are more suited for real world, dynamic semantics. WORKWARE also allow cycles in the classification hierarchy (with cancellation inheritance this can be meaningful [483]). Intensional containment (IC) in CONCEPT D is said to be asymmetric (acyclic). Niinimäki [369] notes that this may not be the case, due to the many kinds of relations that IC represents. The other containment structures in CONCEPT D (properties, relations, and composition), provide a theoretical foundation for the generalised inheritance scheme (reuse along any relations) proposed in this thesis. By allowing reuse along property, part and other relations, semantic holism is achieved in both frameworks.

Summary

This section has shown that the WORKWARE metametamodel compares favourably to existing modelling frameworks. The evolving, ambiguous and incomplete nature of interactive models motivates the differences between our approach and previous research.

The survey also shows that most of the individual techniques applied in WORKWARE are applied and tested in other systems as well.

9.4.2 Process Knowledge Management

WORKWARE's policy-controlled framework for lifetime sharing of reused features is here compared with previous research into process model inheritance.

Layered Policies

The state of the art survey in section 3.4 identified OBLIGATIONS' layered policies [58, 59] as the solution most relevant for interactive process models. Like WORKWARE this system implements multiple cancellation inheritance. However, multiple layers of metaobjects make the framework more complex and rigid (e.g. objects at the same layer cannot inherit from one another). OBLIGATIONS bind templates to instances dynamically according to *surrogate* rules. Rule based, semi-automated reuse is feasible in WORKWARE's approach as well (as reuse decisions), but require a number of features not currently implemented (mapping and parameterisation rules, reuse policy priorities defined by rules). On the other hand, our interactive approach allows greater user control of reuse, which would require *interactive surrogates* in OBLIGATIONS. WORKWARE's aspect-oriented scheme also modularises reuse, e.g. so that users can reuse the work breakdown structure but nothing else from a template.

The PROCESS HANDBOOK

The PROCESS HANDBOOK (PH) project [329] is probably the most comprehensive approach to process knowledge management to date. Here a library of process models is organised in two dimensions, specialisation and decomposition, to support alternative ways of working. The project has developed rules for specialisation [540], composition [47], exception handling [275], and flexible workflow [45]. PH is based on coordination theory [327, 328], and its taxonomy of different coordination mechanisms is heavily utilised [47, 275]. Its language (PIF, cf. section 3.1.1) fit well with EEML. An integration of PH tools and repository into the EXTERNAL infrastructure thus seems feasible.

The PROCESS RECOMBINATOR [47] is an add-on tool to the handbook which provides interactive support for generating new process variants. It lets the users combine alternative specialisations of the sub-activities in a process, and select different coordination mechanisms for the flow dependencies in the model. The approach is based on the existence of a rich repository of process templates (about 5000 in 1999). End user articulation and harvesting of local modifications is not within its scope. Bootstrapping in a new organisation thus requires large initial effort to build critical mass, unlike the low-fidelity early support provided by the EXTERNAL infrastructure. WORKWARE's dynamic, multi-dimensional classification structures could further situate the navigation structures and alternative generation of the handbook. Based on characteristics of the current situation, ad-hoc classification structures could be generated, both increasing the number of combinations (important in early phases) and meeting the current needs more accurately.

Process Model Specialisation

The specialisation rules for the PROCESS HANDBOOK [540] follow closed system semantics, where the model represents everything that can happen. Specialisation is defined as a subset relation on the range of behaviours articulated in the model. Users specialise models by removing alternatives from a large general template. Interactive, open systems follow conventional object oriented semantics for *incremental definition,* where specialised models *add* details to what they inherited. This makes the general models simpler and easier to understand, capturing what the specialised templates have in common, not accumulating every detail. Interestingly, the current handbook seems to follow a more open approach than its original design [540], utilising decomposition and object specialisation more than object removal. WORKWARE's generalised inheritance can define both greatest common denominator (GCD) and least common multiplier (LCM) generalisations (cf. section 3.4.3), as inheritance down and up the specialisation hierarchy, respectively.

Similar transformation rules have been proposed for Petri net inheritance [84, 550] (section 3.4.3). These rules however allow addition of new elements in subclasses. It is a limitation of WORKWARE's cancellation inheritance that automatic substitutability cannot be guaranteed. But for evolving, interactive models, this property is not that important, as users are expected to control to the combination and adaptation of process models anyway. Automatic model assembly from specialised parts is in attractive vision, but unrealistic for knowledge intensive work. It seems more feasible to let users combine model fragments locally in their own projects than expect them to model general fragments in a strict specialisation hierarchy. This is supported by Taivalsaari's assessment that strict inheritance is of limited utility for complex and evolving systems [483], and by case studies that show limitations of strict inheritance for process modelling [298]. For automated processes, however, formal subtyping rules can play a role.

The SOFTWARE PROCESS GENERALISER [227] semi-automatically generates generalised models from a set of particular models. Generalisations can have different degrees of "genericity", defined as the threshold fraction (from 100% to 0%) of individual processes (or subclasses) that must contain each of the elements in the general model. The strict PROCESS HANDBOOK scheme approaches 0% in this range, while strict object-oriented inheritance is at 100% (nothing can be deleted in a subclass) [483]. The GENERALISER opens up for other degrees of template specificity, and is thus well aligned with our framework. Usage experience from software development indicates the usefulness of generalised models with around 50-60% commonality [227], illustrating the potential of situated, interactive decision making over global formal criteria.

Object Oriented Process Model Reuse

E^3 [241, 238] is an object oriented process framework that was favourably evaluated in Chapter 3. A class in E^3 inherits the attributes, methods and associations of its superclass. It is straightforward to define this inheritance rule as a reuse policy in WORKWARE. E^3 places particular emphasis on the definition and management of associations, hence associations are organised in an inheritance hierarchy just like object classes. WORKWARE encodes links as properties or reified as objects. This means that links can be inherited alongside an object, or by itself, just like in E^3. The difference between the two schemes is that E^3 has stricter rules for what kind of inheritance is allowed, e.g. arity may not be changed in the inheriting association, and the classes related by the association must be compatible (substitutable subtypes). Taking into account the discussion above, WORKWARE provide a simpler and more flexible association inheritance scheme, more suited for incomplete and evolving models.

Rule-Driven Process Configuration and Individualisation

PROVE [418, 423] is an object-oriented framework for capturing and disseminating engineering process know-how. From a perspective of engineering processes as "*innovative, individual, dynamic, interdisciplinary, strongly interrelated, strongly parallel, iterative, communication intensive, anticipatory, planning intensive, uncertain, risky etc.*" [422], the approach is based on articulation by process performers, rather than external experts. Reference process models are *individualised* into project specific instance models. Individualisation is partially automated by construction rules, which relate constraints about the application context to process building blocks. This framework is more flexible than the PROCESS HANDBOOK and more aligned with interactive modelling. However, like the handbook it requires detailed process models, and thus would not be sufficient in new domains. WORKWARE's support for articulation by process performers goes beyond PROVE's, e.g. in supporting project specific metamodels and local definition of object attributes, types and relations. PROVE's modelling language also is more complex, utilising semantic holism to a lesser extent. Its construction rules capture the process design rationale through traces of which construction rules were applied as a result of which constraints. This construction meta-process extends the reuse metaprocess and decision modelling proposed in section 5.6.6 with constraint-based advice. These approaches are conceptually and technically well aligned, and could be integrated.

Inheriting and Identifying Exception Handling Alternatives

In ADOME [103], exception conditions and handlers are reused along work and organisational decomposition, resource roles and classification structures. This design motivates WORKWARE's multi-dimensional inheritance framework. Ad-hoc, emergent processes require flexibility for all aspects of the process, not just exceptions, and thus our framework is more generic.

NATURE - Process Model Guidance and Reuse

NATURE [201, 247] supports situated *guidance* and *execution* of requirements engineering processes. Process improvement is anchored in practice by using process traces as input to template modelling. A guidance engine performs pattern matching of the current context against a set of "reusable process chunks" in the repository, suggesting alternatives to the users. The engine thus supports situated composition of processes from templates, and allows deviations from the intended process (ad-hoc decisions). Chunks that together accomplish a common goal are grouped together in *clusters*. Clusters thus represent AND relations between templates, while the bundles in the PROCESS HANDBOOK represent XOR relations. Clusters may be associated with different *modes* that determine the sequence of the chunks, e.g. depth-first or breadth-first. Both could be expressed as reuse decisions in WORKWARE. WORKWARE does not separate descriptive traces from prescriptive models, because an integrated representation of process history, future plans, and the current state is useful for planning, coordination, and learning. NATURE on the other hand emphasises selection among predefined alternatives and capture of underlying rationale. WORKWARE captures the histories of events on modelled objects, but does currently not utilise this information for reuse customisation. Like the PROCESS RECOMBINATOR and PROVE's individualisation support, NATURE offers extended support for automated reuse compared to WORKWARE's meta-process with interactively resolved reuse decisions. On the other hand, poor support for end user articulation of new ways of working is a common weakness of these other systems. Further work on integrating a process repository into the EXTERNAL infrastructure should explore these approaches.

9.4.3 Multiple Models, Levels and Views

Within the ARIS (*Architecture of Integrated Information Systems*) framework [433, 434], event driven process chains (EPC, cf. section 3.1.1) are used to integrate process engineering and knowledge management, planning and control, workflow enactment, and work item performance. Similar to the EXTERNAL infrastructure, but contrary to much workflow research [237, 528], ARIS uses one modelling language at all levels, bridging the gap between business process engineering and workflow enactment. Some proposals for UML 2 [145, 342] similarly advocate the use of a simple, integrating core model rather than the current separation of primitives by diagram types, claiming that this is needed for tool interoperability and activation beyond model editing and visualisation.

Weske et al. [528] advocate a separation between business process (analysis) and workflow (design) models, because the two focus on different aspects. At the same time their case studies report lack of integration between organisational and technical aspects as an important problem. Of their six cases, two decided not to use a workflow management system at all due to severe difficulties. Three were still in the design or testing phase, all with severe delays, while the sixth, which used SAP R/3 (ARIS), was the only one ready for operation. The approach to separate analysis and design is thus not supported by the empirical data. An integrating scheme that supports different views of a common process model for analysis and design, combines specialised support with integration of organisational and technical aspects. Although fully automatic translation from analysis to design seldom is feasible [528], interactive translation, where some steps are automated and some performed manually, is useful. Manual mappings should be explicitly captured so that they can be maintained when the process evolves [237], cf. section 5.6.5. A single modelling language with multiple views, fulfils these requirements [385]. WORKWARE's flexible metametamodel allows multiple levels and views to be added when needed, with new properties, relations and classification structures. Compared to ARIS, our approach is simpler because instances defines structure, properties and behaviour at all levels of abstraction.

9.4.4 Composition of Process Model Fragments

Technically, the major contribution to process knowledge management in this thesis is the inheritance framework and the metametamodel. Composition of process models is enabled by the interfaces (connectors and resource roles) developed for the predecessor APM [90, 91]. Our language has preserved most of these features. WORKWARE extends APM with semantic holism. This implies that a richer automatic adaptation of model fragments can take place when they are put together. For instance, derived properties implemented by reuse policies may cause each fragment to be customised according to the context in which it is placed. The Open Process Components (OPC) [175] prototype allows locally encapsulated semantics of each component. WORKWARE by default takes an open systems approach, in that every component can see every detail of every other component. Access control policies may however be defined to ensure encapsulation and partial sharing, e.g. between the process components performed by different partners in a virtual enterprise. Compared to OPC, WORKWARE's strengths include simpler models and language, e.g. propagation of reused features instead of delegation, as discussed above.

Warboys et al. [515, 516] have developed protocols for interaction between related tasks. They discovered that coordination constituted an important fraction (up to 50%) of typical process models, and sought to isolate coordination into *connectors* separate from the content of the work. Recognising that *"an effective architecture needs to deal as much with connectors as the components being connected"* [515], their approach enables reuse through re-composition of tasks and connectors. Coming from the software process community, Warboys et al. follow the closed system assumption, and compile the process *code*, thus limiting runtime flexibility. On the other hand, decision connectors in WORKWARE would also benefit from richer interaction protocols, e.g. for rule driven coordination of concurrent tasks.

9.4.5 Other Process Model Reuse Techniques

The multi-dimensional process models of the MOBILE prototype [213, 236] motivated the separation of reusable features into aspects in WORKWARE's reuse framework. While MOBILE supports composition of models for each aspect, the semantic holism of WORKWARE captures inter-aspect dependencies better. For instance, the organisational structure for a process may be reused along the work breakdown structure (the functional aspect), and the operational aspect (tools) may be derived from the informational aspect (which data are manipulated). Through interactive enactment, WORKWARE allows execution of processes even though not all the aspects are fully predefined. Other relevant techniques for process model reuse include case-based reasoning [179], and patterns. Configuration management and version control of model templates is another feature not dealt with in this thesis. However, APM [90] outlines an approach to these problems that can be applied to WORKWARE as well.

9.5 Evaluation Summary

This part of the thesis has reported on three different forms of evaluation:

- The implementation of modelling language, metametamodel, and activation semantics in the WORKWARE prototype and the EXTERNAL infrastructure demonstrates that the proposed concepts and designs are feasible.
- The application of these prototypes in case studies shows general usability, and examples from process models developed by users indicate that the more specific techniques are applicable. Experience from the cases also demonstrates the relevance of the requirements presented in Chapter 2.
- Finally, detailed comparative analysis of WORKWARE with existing solutions for workflow modelling, enactment and reuse demonstrate the originality of the approach.

However, the evaluation has also identified a number of limitations to be addressed in further research and development. Some elements of the design, e.g. customisable enactment and reuse policies, have not been fully implemented, and in a number of areas further studies are needed to examine the relative merits of competing approaches. Usage experience also shows the need for non-technical contributions in modelling methodologies, and organisational measures to establish and sustain user participation in articulating work processes.

Chapter 10
Conclusions and Further Work

The root problem of this work was to design a model-driven information system for planning, coordination, cooperation, management and performance of knowledge intensive project work. Previous research in workflow management, conceptual modelling, social and organisational learning directed our efforts towards designing an open, transparent system. Here users actively participate by articulating the plans of their work and by interpreting and elaborating these plans in the situations that arise. In previous work, researchers have tried to improve communication and coordination in ambiguous, uncertain, situated and contested social environments with formal, precise models. We have taken the opposite approach. Rather than bringing the language of computers to human interaction, interactive modelling requires that computerised models approximate human languages. Rather than treating processes like software [389], we treat software as interactive processes. This leads to three research objectives:

- *Articulation*: Develop modelling languages that are simple, flexible, and user-oriented, that allow varying degrees of specificity.
- *Activation*: Develop a process support system that activates user-oriented, evolving and incomplete models, providing useful, contextual, customisable functionality.
- *Reuse*: Develop a framework that enables interplay of local modifications and process improvement, anchoring knowledge management in practice and experience.

Seeking to frame the problem more precisely as a set of requirements, a literature study explored the organisational contexts in which such systems should function. Theories concerning social construction of reality, reflective practice and communities of practice helped us understand that ambiguity is necessary for learning, communication, coordination and the construction of shared understanding. The unavoidable gap between the complexities of practice and any process representation was also highlighted. Case studies showed that there are barriers to articulation, but also that end user participation in workflow management, process modelling, knowledge management, and information systems development, has been successfully achieved in a wide range of industries. The approach should thus be feasible.

10.1 Contributions

Following Schön's [445] agenda for reflective research, this thesis first of all contributes to *repertoire building*. Interactive models are an exemplary problem frame and solution that has received little attention within information systems engineering. We have also looked at the differences between conventional IS development, participatory design, and interactive model construction, contributing to *frame analysis* of how designers attack problems. The contributions of this thesis fall into different categories [166], including general approaches, new explanations and perspectives, new functionality and new implementation techniques. Such a wide variety can be expected from engineering projects that span from understanding user organisations and their requirements, by way of technological challenges and methods, to actual implementation. The drawback of such vertical studies is of course their limited horizontal breadth. It is beyond a realistic scope to investigate all relevant areas in all levels. Consequently, while aspects of these contributions have been proposed in other areas, they are new in the areas of interactive models and workflow management, and their application in these particular areas point to new considerations and combinations, bringing a clearer understanding of their benefits and limitations.

10.1.1 General Approach: Interactive Models

Conventional IS, built for low maintenance and long life span, often become obstacles to organisational change and innovation. They become too rigid for dynamic work and too general to meet the unique local challenges. Consequently, only a minor portion of the functionality is ever used, as powerful and specialised features encode assumptions about the usage environment that are seldom fulfilled. Current techniques for building more flexible systems "*include prototyping, end user development and open systems connectivity. But these are inadequate because they are not connected through a coherent framework that focuses on the emergent character of organisations*" [497]. This thesis argues that interactive models can function as such a framework. By facilitating end user participation in modelling, and allowing users to contribute also to the interpretation of models, a flexible, evolving system is feasible. The behaviour supported by the system is no longer fixed to what is prescribed in program code. An open, interactive architecture allows customisable software services to be added when needed. The models provide a context for software services, defined in user- and domain-oriented terms.

IS engineering relies heavily on conceptual modelling [101]. This thesis contributes with an elucidation of the concept of *interactive models*. Interaction was introduced by Wegner et al. as a theoretical framework for computing [523] and conceptual modelling [524]. This thesis points to interactive models as a *design metaphor* for flexible, user-friendly information systems. It defines interactive models as models that are available to end users and that control the behaviour of the system, and point to the essential duality of articulation and activation. The project further uncovered interactive modelling requirements and challenges, and identified suitable modelling techniques. To my knowledge, no such practical exploration of interactive models has previously been carried out. The interaction framework is highly contested by some scientists because it does not really say anything new, theoretically that is. What it provides is a new perspective, a different way of framing, describing, analysing and solving problems. Demonstrating its practical utility in design is thus essential.

10.1.2 Articulation

In order to facilitate process modelling by end users, simple, user-oriented, and flexible languages are needed. This thesis identifies a number of techniques for this purpose.

Semantic Holism

Semantic holism expresses meaning through combined interpretation of all the elements in a model. Popular modelling languages today provide limited degrees of holism, and rely mostly on atomic semantics, where the meaning of a model element is defined by that element alone. While the concept was introduced elsewhere [263], this work contributes with the identification of holistic modelling techniques like

- *Property modelling*, where properties can be dynamically attached to object instances,
- *Constellation modelling*, where meaning is expressed by constellations of objects, each affecting the interpretation of the others, and
- *Context-sensitive semantics*, where model elements change meaning according to the context where they are placed.

The modelling framework designed and implemented in WORKWARE demonstrates how these techniques result in simpler, more flexible language, meeting requirements of multiple views, domain specificity, and evolution of models, semantics, and modelling languages. A study of related work show that these techniques have a wide range of application.

Explicit Decisions Enable Interactive Activation

Decisions control the interpretation of the model. By explicitly articulating decisions as model elements, manual and automatic activation can be integrated. Many elements in existing languages implicitly capture decisions, so conventional modelling languages can be transformed into interactive ones. Decisions also simplified the process modelling language APM. In WORKWARE, the location of decision objects defines decision-making authority and responsibility. In addition to workflow and scheduling, reuse and resource allocation decisions have been modelled. Case studies show that these constructs are useful. This technique helps us view *modelling as a process*, where meaning is negotiated.

Feasibility of Articulation by End Users

Some researchers [35, 213] claim that end users cannot be trusted to change workflow models correctly. This thesis refutes this claim by identifying a number of case studies where end users did change such models. Another explanation of this phenomenon is thus offered, that the complexity of changing *models at the class level*, where you have to take into account all past, present and ongoing processes that follow the same model, prevents end user modelling in adaptive workflow. Consequently, this problem need not apply to emergent workflow, where users model their unique project. Our usage experience supports this claim.

10.1.3 Activation

The activation of interactive models to support knowledge work has generated some new design perspectives. These contributions primarily deal with the behaviour semantics of interactive models and the information systems architecture.

Interactive Activation

Interactive activation was introduced in order to allow incomplete and evolving models. In this scheme users and software components cooperate in interpreting the models in the situations that arise. Software can activate fully articulated model fragments, while human involvement is needed for ambiguous parts. Humans can also override prescribed behaviour when faced with unforeseen exceptions, through in situ articulation. This is exemplified by interactive workflow enactment, which allows workflow instance models to evolve past different degrees of preciseness and detail. Interactive enactment has been implemented and used in case studies. Compared to related work, it offers a simpler, more comprehensible model, since in an open system, we need not predefine every potential enactment pattern.

Holistic Activation

The realisation that semantic holism is a core feature in interactive activation, is another contribution. When the interpretation of model elements depends on the current states of surrounding elements, richer, more situated functionality is enabled. The activation emerges from the local interpretation of individual elements, better matching the contingencies of work. The ability of holistic activation to handle more complex scenarios than conventional atomic solutions has been demonstrated.

Tailoring with User-Oriented Concepts

With an interactive model, domain and user-oriented concepts are applied for tailoring the system. Most tailorable languages contain system-oriented primitives, allowing users to assemble components and combine features. In WORKWARE, interactive models are utilised for *contextualising* the system. System-oriented customisation *policies* are introduced only where such a solution is more simple and straightforward. Semantic holism enables *implicit* reconfiguration of the activation, controlled by the current state of the model. Policies allow explicit reconfiguration, while interactive models facilitate implicit, partial and dynamic reconfiguration. WORKWARE integrates these techniques.

Integrating Multiple Model Interactors and Aspects

A model interactor is a software component that interactively activates parts of a model. This thesis shows how multiple interactors, offering complementary interpretations of the same model, can be combined in an interactive architecture. Property modelling and semantic holism facilitate integration of multiple perspectives without complicating the language. This is demonstrated by the WORKWARE architecture, which combines workflow and groupware components to provide richer coordination functionality.

Semantic Completeness and Correctness Revisited

With interactive model activation, the notions of model completeness and correctness are revisited. Rather than dealing with formal semantics, we emphasise real world semantics, what the model says about the real objects it refers to. An interactive model is complete only when it is no longer in use, when all the work it represents has been performed. Only then is it no longer subject to the interplay of articulation and activation. No more exceptions and further planning need to be articulated. Similarly, process history is the only part of the model that is fixed. Formal semantic correctness is no longer a prerequisite for activation, as users can handle ambiguities, incompleteness and inconsistencies.

10.1.4 Interactive and Emergent Workflow

The difference between adaptive (dynamic) and interactive, emergent workflow is described, clearing some of the conceptual confusion that has prevailed in this area [45, 46, 274, 276]. While adaptive systems aim at making top-down workflow control more powerful, emergent workflow supports bottom-up articulation of work, enabling employee empowerment and partial group autonomy. This conceptualisation also points out new directions for further research, outlined below.

10.1.5 Reuse and Process Knowledge Management

In this thesis, reuse of process models is viewed as an interactive process. The aim is not to provide complete generic models but to facilitate articulation and composition of template fragments by the process participants. Most KM strategies take a post-hoc documentation approach, asking "what have we learned from this project". Their goal is to make components reusable at the time of their production. We argue that *need-driven reuse* asking, "what past experience is relevant for this situation", better matches the open, situated nature of cooperative knowledge work

Incremental Classification with Instances as Primary Objects

In order to allow local modifications, WORKWARE defines all structure, behaviour and other features of model elements at the instance level. Any model element can be reused. Classification structures are dynamically constructed to meet local, evolving requirements, they are not an inherent part of the object definition. Their main purpose is to help process participants identify relevant past experience and model fragments suitable for the current situation. Experience shows that meta-levels are among the most difficult modelling concepts to grasp [298], so this is an important simplification.

Generalising the Concept of Inheritance

Inheritance conventionally couples reuse to specialisation. In our framework, reuse may also take place along (de)composition, work flows, and other relations. The realisation that some model aspects should propagate along other relations was triggered by specific user needs in our projects. A detailed design shows how this scheme can be implemented without destroying the one-to-one domain to model correspondence that instance modelling facilitates.

Reuse Policies

Policies control the dynamic binding of behaviour rules to model elements, e.g. for enactment, user interfaces and access control. Reuse policies define how reusable aspects are propagated along modelled relations, allowing user organisations to customise the semantics of inheritance. Basic reuse policies have been defined based on the requirements from different case studies. In e.g. user interface customisation and access control, several reuse policies have been implemented.

10.2 Limitations and Directions for Further Research

According to reflective practice, engineering research generates new problems as well as solutions. This work has implications for further research into usage, evaluation, methodologies, design and implementation of model-driven process management.

10.2.1 Validation

This thesis concentrates on the technical component of a sociotechnical information system. Organisational, social and human sciences have been surveyed in order to establish requirements for the IS, but long-term studies of interactive models in real organisations have not been carried out. Following the discussion in Appendix A, we propose case studies instead of controlled experiments to capture the complex and incompletely understood realities of new work practice. Subjective storytelling and interpretive fieldwork are more appropriate than positivist studies with formalised hypotheses. From the point of view of an IS designer, these questions should be investigated:

- How can we motivate initial and sustained user participation in articulating and sharing interactive models?
- What are the barriers to user participation and how can we overcome them?
- To what extent can we expect end user innovation in model articulation, process improvement and language evolution?
- What are the characteristics of organisations that are able to capitalise on interactive models as opposed to those that are not?
- How does the introduction of interactive modelling influence cooperation, communication and culture of different organisations? In which cases is openness increased and when do people withdraw from articulation and sharing of work descriptions?
- What aspects of work are suited for articulation in different organisational settings and what aspects are better handled informally? For what aspects will the gap between models and reality be too wide for models to be useful? What level of detail and preciseness is most useful?
- Are project plans as process models the most useful kind of interactive models, or are other aspects, e.g. product or information models, more suitable in some settings?
- What kinds of model-driven functionality are most useful in different situations?
- What roles must be performed in the introduction and use of interactive models? When do we need system experts that help users to customise their solutions, and process experts that facilitate articulation work? How can the software system best accommodate these roles?
- How can interactive models be utilised in organisational change processes?

10.2.2 Methodologies

Methodologies for interactive modelling depend closely on the social and organisational settings where models and systems are applied. Although the EXTERNAL project has developed a general methodology [286], customised methods are needed for:

- Integrating *project management*, monitoring and resource control methods with interactive models,
- *Solution provision*, enabling IS consultants to help organisations customise and extend their interactive modelling systems, emphasising end user participation.
- *Business consulting*, introducing interactive models as a way of transforming organisational structures, processes and cultures.
- *Modelling by novice users*, a simplified tutorial.

These and other methodologies must go beyond user guides, also designing learning material and workshop formats. The *modelling conference* approach [181, 182] is an interesting starting point. It has been successfully applied to process models, but needs extensions to sustain participation in articulation after the workshops are closed. Finally, interactive models offer a unique opportunity to *operationalise methodologies as processes*. By utilising the process knowledge management features, we may grow methodologies from the practical experiences of project managers, consultants and users. A duality of community building and repository building is needed.

10.2.3 Design

Design ideas have been put forward as the main contributions of this thesis. Current development of WORKWARE is directed at further situating the interpretation of models, without adding too much complexity to the language. This includes extending the automatic support of decisions to include constraints and operational rules, and to utilise temporal considerations (milestones, timers, delays etc.). These model elements must currently be manually activated. Other approaches that can help to further situate the interpretation include explicit representation of vagueness, uncertainty [215] and changeability [164] of model elements. Other promising extensions and refinements to the architecture and modelling framework include:

- An open model editor that fully utilises semantic holism and instance modelling. METIS does not adequately support properties, constellations or contextual semantics.
- Improved process visualisations, e.g. to manage and coordinate large projects. The SUPREME visualisation tool [253] is a starting point.
- Textual model explanations [205] and textual articulation should be investigated. Text and visual models should be integrated in interactive hypermedia.
- Integration with organisational resource planning systems, utilising competence skill profiles and training plans to allocate the most suitable people to each project.
- Integration with document and information management systems, e.g. utilising process structures for information retrieval.
- A more dynamic and fine-grained utilisation of available software components, e.g. through alignment with web services and business object standards.
- Integration of time and project management functionality, utilising up to date representations of the current state of the project.
- The capability to refer to past and future states of the system, and express relations between them [24] could further contextualise model interpretation.
- Improved conversation and negotiation support, integrated with standard email so that users needn't install yet another communication tool.
- Support for process analysis, metrics and assessment methods, including simulations that take the process history into account, and system dynamic models that capture holistic interdependencies among tasks, products and resources.
- Improved support for model-driven task automation, utilising detailed enactment rules. The current language does provide the expressiveness of workflow management systems, but detailed automation is not adequately supported.
- Guiding and warning systems that monitor the enactment process and provide feedback to users about rule violations, exceptions etc. Process models should be utilised to decide who needs to know about which situations. Deontic rules, as integrated in APM, would help us approach this objective.
- Interactive multi-level and multi-user undo functionality for intertwined model articulation and activation. Transaction support should also be added.
- Template management, configuration, versioning and change control. Interactive meta-process approaches to these problems seem promising.
- Utilising interactive enactment and other WORKWARE components in existing IS infrastructures that were not originally designed for interactive models.

Most of these designs deal with developing new model interactors. The principles of semantic holism at the model, interactor and user levels should guide these designs, in order to provide simple and flexible solutions.

Process Knowledge Management

Process model management and reuse is the area where the current designs most needs extensions. In particular, WORKWARE needs a model repository that allows fine-grained access to model objects with acceptable performance.

- Experience from search tools, classification systems and library science should be utilised to support the identification and selection of suitable model templates [223].
- Process model *patterns* should be investigated as an approach to capture the suitable usage context for different models, and for defining general, reusable structures that can be contextualised (via inheritance) into local models.
- The application of *case based reasoning* in a semi-automated template selection interactor should also be looked into.
- *Storytelling* is a key process for informal knowledge dissemination in organisations. Case studies show that this can complement explicit organisational procedures.

Knowledge management in general can benefit from the interactive models perspective, in that organisational improvement based on explicit models can be integrated with project and group level models that capture situated experience. Interaction is currently replacing automation in several KM tools.

10.2.4 Implementation

WORKWARE is a prototype, and although it has survived 5-6 years of experimentation, a re-implementation is needed for full commercial utilisation. The challenges involved in implementing a commercial product include:

- Improved performance in model data management. Experiments are needed to verify e.g. that WORKWARE's metametamodel can be handled by a relational database system. Transaction support and undo-services are also needed.
- Distribution of functionality and processes to different servers should be supported.
- The user interface should better utilise XML and stylesheets, and a user interface for defining new forms and components should be developed.
- Integration with external tools and web services must be improved.

Tinella has started to attack some of these challenges in Computas UEPS [491].

10.2.5 Generalisation

While this thesis has concentrated on interactive models of work processes, we have also briefly looked into interactive use of product models, organisation structures, ontologies etc. Investigating the applicability of the contributions from this thesis to other domains, is thus also a major challenge for further research. The application of interactive modelling during systems development should also be explored. Interactive model execution and translation are promising validation techniques. Interactively executable prototypes should be more easy to generate than fully automated ones. As we have seen, interactive principles can contribute to simplify systems modelling in UML. After all, the core problems of this thesis are quite general: Motivating end user participation, providing simple, transparent systems that the users can control and not be alienated by, supporting the essential activities of knowledge work as well as automating its routine parts. The common approach of bringing the language of computers to human interaction, should be replaced by research aiming to make computer languages more human.

Appendix A
Research Methodology

Research aims to produce knowledge that is original, grounded, tested, systematised into theories and practically useful. These objectives have given rise to a number of different methods [545]:

- Science, developing theories that give rise to hypotheses, which are then tested.
- Engineering, developing, testing and refining solutions to problems.
- Empirical methods, where data are collected to verify hypotheses, but unlike in science the hypotheses are not derived from theory.
- Analytical methods (or formal science [81]), where a formal theory is deduced from axioms by application of the laws of logic [208].

This is an engineering thesis. Empirically oriented scientists currently challenge the position of engineering research in software engineering and computer science (CS) [463, 490, 545]. This appendix argues that the following assumption is reasonable:

- There is a need for engineering research. The fundamental problems of software development cannot be solved by science alone.

The core of this argument is a practitioner's perspective, emphasising *management of complexity* as the key challenge [521].

A.1 Scientific Research

It is the method that distinguishes science and scientific knowledge from ordinary practice and knowledge. Scientific knowledge can be considered "truer" because the scientific method tests it, identifies its shortcomings and corrects them [81]. A coarse depiction of the scientific method is given in Figure 82. It shows the generative characteristics of the process, that new knowledge and new problems are both starting points and outcomes. The core activities include hypothesis development, testing and evaluation. Though not all research disciplines assign equal importance to each step in this process, it seems to be a suitable framework for discussion.

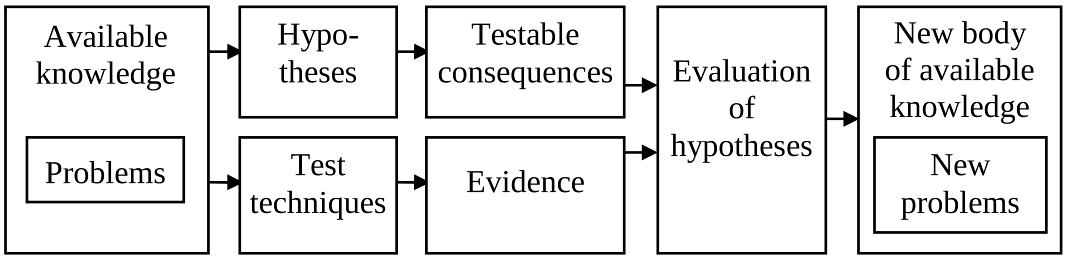


Figure 82. Scientific method [81].

A.1.1 Concepts

Scientific knowledge takes the form of symbolic, conceptual representations. *Concepts* (units of thought) are thus the primary medium of science [81]. The scientific process deal with formation, elaboration and manipulation of concepts. Empirically, concepts are tools for identifying and grouping objects. Concepts can have a linguistic representation as *terms* in a language, and refer to objects and properties in the physical world, as summarised in the table below.

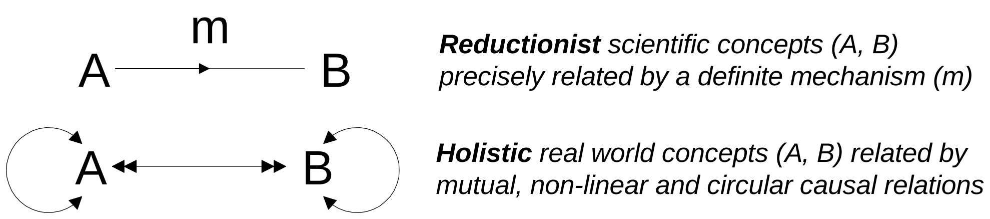
Level	Unit	Links
Linguistic	'Term'	Term *designates* concept
Conceptual	"Concept"	Concept *references* thing
Physical	Thing, property, phenomenon	

Table 13. Semiotic levels.

The meaning of a concept is determined by its *reference* set, the physical and ideational objects it refers to, and its *intension*. The intension specifies the properties that apply to the concept [81]. Two concepts are synonyms if both their reference set and their intension are equal. Concepts can be categorised according to *logical strength* (ability to express facts precisely) into *individual* (definite or indefinite), *class*, *relation* (comparative or not), and *quantitative* concepts.

A.1.2 Holistic Reality and Formal Theories

Due to our limited capacity to observe what goes on in the world, reality will always seem holistic [524]: Causal relations appear as mutual, circular and non-linear (every observable cause has an indefinite number of effects) [194, 447]. This is illustrated in Figure 83. As an example, let us look at the apparently linear causality that "users participating in modelling their work" (A) increases "visibility of work" (B). Organisational factors and software tools, e.g. access control mechanisms, may however prevent this straightforward causality. The mere fact that work becomes visible may also bring counter-measures (users withdraw from modelling because they fear that increased explication will lead to increased organisational control and micro-management), or motivate participation (sharing your experience brings prestige and self-actualisation). Network effects and peer pressure among colleagues, are also known to influence user participation. Such descriptions of complex and contingent organisational realities are incomplete and generalised. More issues regarding user participation and visibility of work is elaborated in Chapter 2. No model of the real world can ever be complete in the sense that it captures everything that may be relevant for a phenomenon.



m
A
B
Reductionist *scientific concepts (A, B) precisely related by a definite mechanism (m)*
A
B
Holistic *real world concepts (A, B) related by mutual, non-linear and circular causal relations*

Figure 83. Holistic and scientific concepts.

Science on the other hand, tries to see the world from the eyes of an omni-present, non-interfering, objective observer [524]. If scientific hypotheses are to be testable, all variables cannot be holistically related to all other variables [81]. Thus, hypotheses are generally formulated with linear cause-effect relations, as in conventional logic [208]. This implies that scientific concepts are fundamentally different from ordinary concepts, as clarified in Figure 83. Scientific concepts can be linearly related, but they cannot at the same time be directly observable, because no experiment can ever assume the position of an objective, omnipotent observer. "*In real situations, the relevant variables are seldom adequately known and precisely controlled. Real situations are much too complex for this, and effective action is much too strongly urged to permit a detailed study*" [82]. Through controlled experiments in a laboratory we can observe phenomenon that approximates the scientific concepts, but they can never be observed directly. Even in the most thoroughly controlled experiments, unknown factors such as the limitations of measuring devices can cause non-determinism. This implies that scientific concepts and theories never capture the *whole* of the real world objects they refer to, but only a subset of its characteristics [81]. Hence, practice-oriented social scientists have used concepts of duality rather than classes (dichotomies) in their theory building [466, 527]. The tension between holistic and scientific worldviews [38, 194] is evident in most of the areas that are relevant for information systems engineering. The disciplines of experimental research reflect different approaches to this issue:

- Empirical research deals with observable concepts, and in some cases even requires that all concepts should defined by the way they are measured (operationalism).
- Factual science builds systems of reductionist theories, but tests them in the laboratory (and to a limited extent in the real world). Scientists must thus derive observable consequences of their theories, as shown in Figure 82.
- Formal science builds systems of logical concepts, interrelated in a linear way. The laws of logic are universal. Because no empirical data can contradict logically proven facts, formal models cannot say anything about the real world.
- Engineering attacks a holistic problem situation. Due to the limited scope of any theoretical principle, personal skills, intuition, social dialogue, folk wisdom, rules of thumb, and analogies are also valued techniques [111].

From this perspective, engineering *research* has to find a suitable balance, attacking the holistic problem at hand, while at the same time documenting and validating the solution in manner which approaches science. In the following, we investigate how this balance is kept in the research disciplines most relevant to this thesis. First, we briefly discuss the potential for interdisciplinary research based on scientific or holistic concepts.

A.1.3 Interdisciplinary and Multidisciplinary Concepts

Interdisciplinary research and practice require some degree of shared language, concepts and meaning across disciplines. Science and practice require different properties of a language, as illustrated above. Scientific concepts are interconnected in systematic theories, indirectly validated by experiments. Experiments require operational concepts, defined by how they can be tested. Although operationalism can be criticised for confusing meaning with testability [81], operational concepts and hypotheses permeate both scientific and non-scientific empirical research. Both theoretical (formal) and operational concepts aspire to be unambiguous and objective, to act as *immutable mobiles* [300], incapable of being translated into something else. Immutable mobiles, aimed at preserving paradigms, reflect the controlling aspects of the term 'discipline'. Scientific concepts thus lead to *multidisciplinary* research where each community has its own local, well-defined vocabulary, and there is no common language.

Practice-oriented concepts have a different role. They are to facilitate communication and participation in an ongoing negotiation of meaning across disciplines and communities [527]. Such concepts act as *boundary objects* [462]. They should maintain their identity across disciplines while having the potential for becoming differently structured by use inside each community. Ambiguous concepts invite participation in the process of negotiating their meaning [527], hence practical, interdisciplinary concepts should act as *mutable mobiles*. The role of metaphors in innovation [370] is an example of conceptual boundary objects. Holistic concepts thus allow *interdisciplinary* research with a partially shared language. Concepts may however have specialised local meaning within each community, reflecting particular problems, theories and methods. While scientific paradigms [293] are maintained by immutable mobiles, creative and revolutionary science needs more dynamic concepts. Thus, Bunge [81] notes that the conventional rule that all concepts should be defined prior to scientific study, should be replaced by a requirement that all the objects of study are identified (like boundary objects). Scientific knowledge is conceptual, so scientific progress requires conceptual evolution.

A.2 Information Systems Engineering

The main objective of information systems engineering is to improve the methods and practice of system development. Research in this field thus utilises a mix of methods from mathematics, computing, social and human sciences, but also a wide range of technological design principles, paradigms and processes developed through industrial practice. The figure below gives an overview of research paradigms. It follows the categorisation above, but adds a separation between interpretive fieldwork and controlled experiments. This is a key separation in social sciences [273]. Interpretive, qualitative methods like ethnography and case studies have also been accepted in information systems research [25]. Engineering combines *interpretive* understandings of reality with *normative* propositions to further professional development [337].

Human, social, and organisational focus	**Empirical science**	**Interpretive fieldwork**
Technical solution focus	**Computer science**	**Engineering**
	Well-defined problems (reductionist, positivist)	Wicked problems (holistic, interpretivist)

Figure 84. An overview of research relevant for information systems.

Investigating the role of knowledge in software development, Robillard [413] notes that formal methods are mainly applicable to well-defined (tame) problems. The notion of "wicked problems" is attributed to Horst Rittel [482]. To such problems, no exact solution exists, rather a *satisficing* [334] compromise of conflicting goals must be sought.

Wicked problems have no definite, generally agreed upon formulation, rather an understanding of the problem is developed alongside the solution [112]. Every wicked problem is unique and novel; requires complex judgement about the suitable level of abstraction; has no objective measure of success or completion, no right or wrong solutions; has no given alternatives; requires iterative problem solving; and often has strong moral, political or professional dimensions [455].

A.2.1 Engineering Research

While the canonical objective of science is the generation of new, tested knowledge, technology's main objective is the design of solutions (technological artefacts) to a problem. "*The questioning of distinctly technological ideas has a different content than the questioning of scientific ideas. The assumption among technologists is not that the technological ideas are true but that they work, and that the works to which they give rise are good or useful*" [353].

Reflective Practice

The often-observed gulf between theory and practice [19, 387, 445] and the conceptual and methodological differences outlined above, indicate that engineers learn more from colleagues and from interacting with the problem situation, than they do from science and scientists. Even a philosopher of science states that "*a practical man is one who acts in obeyance to decisions taken in the light of the best technological knowledge - not scientific knowledge because most of this is too remote or even irrelevant to practice*" [82]. "*You can't research or test your way to good design; you can only design your way there*" [376]. Thus, there are calls for engineering education rooted in *design*, rather than mathematics and science [403, 445]. A survey of experienced software engineers [306], identified methods, design, processes, professionalism and ethics as the most important knowledge, while mathematics, science and hardware were ranked lowest.

Schön [445] gives an excellent account of how engineers think and act. He describes professional work as a *reflective dialogue with the problem situation*, where new designs (solution hypotheses) are continually developed and tested. To attack wicked problems, *problem setting* (framing, analysis, understanding) and *problem solving* must be intertwined. Each new proposal may contribute to solving the current problems, but also to building an understanding of the problem, trigger new perspectives and ideas for solution. Social dialogue, rules of thumb, tacit and personal knowledge are paramount for testing and assessment. Subjective valuations of aesthetics and harmony among the parts of the solution are also important [273]. The technological process is thus less explicit and formal than the scientific method (Figure 82), and its problem-hypotheses-testing-evaluation cycles are more rapid. The creative phases of science, those concerned with hypothesis generation, seem to have a lot in common with reflective practice. Idea formation and hypothesis creation are the most overlooked aspects of the scientific process [81], and in some empirical disciplines, speculation is disregarded.

Repertoires of Action

Crucial, both for framing the problem and for generating new designs, are the *repertoires of action* that the designers have learned [445]. These repertoires contain exemplary solutions, approaches and perspectives gained through practice, education and interaction with other practitioners. In software engineering, *design patterns* [173] explicitly represent action repertoire elements. Reflective practice creates new knowledge not by giving rise to general principles, but by adding to the practitioners' repertoire.

Reflective Research

From this perspective, Schön identifies four areas for practice-oriented *reflective research* [445]:

- Frame analysis, concerned with which ways of framing problems, which paradigms [293] of reality construction [43] that dominate a profession, and the discussion of alternative frames, values and approaches within the professional community.
- Repertoire building, accumulating and describing exemplary solutions in ways useful to reflection in action. Case methods in business and medical education exemplify this form.
- Fundamental methods of inquiry, the theories that practitioners apply to make sense of new situations, which commonly do not fit the theories in an objective way.
- Research on the process of reflection in action involves inquiry into learning, cognitive, affective and group dynamic aspects.

Following this description, it makes sense to view engineering research as complementary to and interrelated with the factual sciences. Engineering research takes a more pragmatic approach. Its main criteria are utility and usability of results, not that the claims are scientifically tested. The main objective is to contribute to the repertoire of action of a profession, not to deliver new scientific theories.

Science and Engineering

In debates between scientists and engineers, it is commonly stated that a mature professional discipline must be firmly based on science, rather than tacit skills, intuition and individual creativity [463, 490, 545]. Stable rules should replace ad-hoc solutions. Scientists ask "*are we ready to uncover our eyes and look at the software problem from a scientific standpoint?*" pointing out that "*without measures from repeatable experiments, software is not science*" [302]. Knowledge that is not scientifically based is thus disregarded. Some even view engineering as applied science [514], and see scientific inquiry as *the* path towards establishing software development as an engineering discipline.

The discussion above challenges this view. Studies of design, innovation and invention [370, 398, 503], show that science offers little assistance in managing complexity and generating new ideas. Distinctly technological concepts have developed, not rooted in underlying sciences, including "machine", "efficiency", "information", and "network theory" [353]. The development of new technology is an important object for diverse fields such as philosophy [82, 353], history [398], economics [503], sociology [51, 300], and organisational science [370]. It is thus not a well-defined area of analysis.

Mitcham [353] investigates the philosophy of technology. It is found to deal with ethics and practice, while the philosophy of science is more associated with logic and epistemology. The etymological roots of 'technology' are found in ancient Greek [353], combining *techne* (art, craft, skill) and *logos* (words, speech, reason). *Techne* is separated from habits based on just experience (*empeiria*) in that they are explicitly reasoned about. To Plato, *techne* and *episteme* (systematic, scientific knowledge) are closely related. *Techne* in the classical understanding is "*fundamentally oriented toward particulars instead of toward the efficient production of many things*" [353]. The origin of technology is thus more concerned with design and problem solving than rationalising mass production, and a clearly separate concept from science and empiricism. "*There is at the heart of technical activity*" "*an irreducible nonlogical component*" [353].

The sociological study of technology [51] emphasises social construction of innovative concepts and artefacts, moving away from the individual inventor, the genius, as the central explanatory concept. Historical studies seem to justify this perspective [398]. In current practice, it is argued that science and engineering have become intermixed, that both disciplines discover as well as apply knowledge. The division between science and technology is seen as a social construction. *Interpretative flexibility* of empirical data [51] is emphasised as an important aspect of revolutionary, paradigm-shifting science [293]. Normal science on the other hand, seeks to minimise the interpretative flexibility of empirical data. A similar distinction is found in technology, where *revolutionary inventions* cause the construction of new systems, and *conservative inventions* improve components in existing systems.

Some economic studies emphasise the interplay of product and process innovation in technological evolution. In the Abernathy-Utterback model [503], an early *fluid phase*, in which many product designs compete and the level of product innovation is high, is followed by a *transitional phase* where product variety decreases and *dominant designs* emerge, possibly even becoming standardised. Now innovation of manufacturing processes speeds up. When the potential for efficiency and quality improvement is exerted, industries may enter the *specific* (or mature) phase, where cost, volume and capacity matters, and innovation makes small, incremental steps. Empirical analysis of product features becomes feasible and appropriate once a dominant design is established. Clearly, some products of the software industry have entered this phase, e.g. operating systems and relational databases. While workflow systems for automating routine procedures seem to be in the transition phase, flexible workflow management currently is dominated by product innovation.

A.2.2 Organisational Science

Engineering research shares its orientation toward practical problems with organisation and management science [20, 39, 334, 357]. While engineers traditionally have applied knowledge from mathematics and natural sciences, organisational theory is based on human and social sciences. However, this distinction is blurring, e.g. in information and computer science [38, 129]. Information systems and organisational design are interdependent, especially in cooperative and process oriented systems [307, 407]. Because of its root in mathematics, there has been a tendency to relate computing to the natural sciences [490], but for information systems, the relationship with organisation science is more relevant [129].

Case Studies

In order to capture the holistic reality of a situation, case studies of real organisations and groups have a prominent place in organisation science and education [19, 39, 370, 387, 445, 468]. Based on various theoretical perspectives, the development or organisation of one particular company is described. This kind of research is mostly qualitative, since few theories in the social sciences are quantitative. A major problem for case studies is to what extent their findings can be generalised to other organisations, and further to theory [81]. Finding the right case, the "*representative sample of one*" [38], is thus paramount. For these reasons, multiple cases are often combined in a comparative field study, but still a great number of cases are needed to verify even the most low-level empirical hypothesis. From a practical point of view, however, these studies are important because they contribute to the discipline's repertoire of action. Attempts at increasing the scientific content of this work has lead to empirical paradigms that accept as meaningful only what is directly observable in the data and criticise attempts at theory building [81]. This leads to less applicable results.

Action Research and Interference Problems

Another important problem in organisational science, is the interference of the researchers upon the domain of study. *Action research* [25, 38, 307] accepts interference as unavoidable. Interpretive approaches [273] sees interaction between researchers and the subjects as the best way to gain insights into the practice that is studied. Active participation also allows the researchers' hypotheses to be tested. In an overview of the action research approach to information systems, Baskerville [38] discusses how attempts to introduce scientific methods "*too often disconnected theory from reality, making the research results largely irrelevant*". He also points to the lack of generally agreed criteria for evaluating action research as a major problem. These concerns also apply to engineering research, which shares both the intervention and complex problem orientation with action research. Some people classify engineering projects as action research [25]. A core difference between the two however remains, that action research uses action to seek understanding and knowledge, while engineering applies and develops knowledge with action and invention as the ultimate objective [337].

Recognising the situatedness, complexities and contingencies of organised activity, *multi-perspective approaches* that draw from a variety of sciences are also frequently applied [273, 357]. Integrative schemes like *sociotechnical systems* [495] have developed as organisation theoretical concepts, not derived from underlying factual sciences. Such concepts have also been imported into computer science [215].

A.2.3 Computer Science and Software Engineering

Denning [129] gives a structured overview of computer science. He challenges the view that *"what can be efficiently automated"* is the fundamental question of the discipline, arguing that it overlooks the full richness of the human, social and historical contexts which computer science also deals with. Algorithmic thinking, representation of data and knowledge, programming and design are the basic skills of the discipline. Design connects the other skills to the concerns of users. It is a commonly held position that CS should be both engineering and science, and more [128, 171, 210, 316, 539]. The field is interdisciplinary. In addition to the mathematics, engineering and science woven into the discipline itself, it is formed by relationships with library science, management science, economics, medicine, biology, psychology, linguistics, philosophy and other human sciences.

Software Engineering

More than 35 years after the term was first coined, the nature of software engineering still is the subject of debate between practitioners who deal with the complexities of real world systems, and scientists who find answers in formal methods and empirical research [56, 184]. These groups were established as early as the second conference on software engineering in 1969 [151]. Though such debates can be counter-productive, they may also lead to improved understanding, shared values and better research [211]. Positivist theoreticians emphasise that software development should be made rational [151], systematic, disciplined [114], "*automatable and easy*" [184], technically deterministic and empirically testable. Some argue that "*anything other than strictly mathematical terms*" should be forbidden, especially human analogies [111].

Practitioners see the essential challenges [76] in software engineering as wicked problems, requiring reflective practice. There are several strategies for *taming* wicked problems prematurely [112], e.g. locking the problem definition, disallowing changes to requirements; specifying the problem by objective and measurable parameters; treating the problem as just like a previous one, ignoring contradictory evidence; simply declaring the problem to be solved; giving up looking for solutions and just follow orders; or pretending there are just a few alternative solutions, and cast the problem as a selection process. While such strategies are appealing in the short run, they tend to fail [112].

"*Rationalising software processes involves standardising intellectual work, which is historically difficult and most likely counter-productive*" [151]. Nevertheless, research into software process improvement has aimed to "*increase precision and speed by repeating the events, reduce the dependency of experts and generally improve control*" [240]. This illustrates that *"the automatable-and-easy people seem unaware of the significance of what the complexity people have said"* [184]. Terms like 'software factory' and 'empirical software engineering' [490] illustrate that some scientists do not relate to wicked problems.

The Push Towards Empirical Computer Science

That the empirical, positivist perspective is currently gaining ground, is shown by studies of published papers [545], funding agencies' strategies [57], and professional certification schemes [278, 494]. The risk of computer science becoming increasingly theoretical is thus imminent. The tendency to favour basic and technology-oriented research over applied and user-oriented, lead practically inclined researchers to "*toe the line or leave the academy*" [132]. People who argue in favour of more experimental research commonly criticise the poor quality of testing in contemporary publications [490, 545]. To compare the contributions of empirically oriented computer science to the more engineering oriented, is a too wicked problem for scientific study.

Practical utility of results and the interdisciplinary nature of most problems, are reasons for adopting a less empirical approach. Wicked problems should not be attacked with thinking, tools and methods useful only for tame problems [112]. A typical example is to measure conformance with requirements as the criterion for good design [129]. Striving to provide precise evidence, some scientists reduce their problems to what can be tested with the available resources. What they disregard are the wicked, not articulated and poorly understood aspects, the very parts of the problem most challenging in practice. Practitioners' response to such research will often be *"but my project is different"* which *"really is a valid response"* [185]. *"When we are concerned with matters of the real world, theoretically-based critiques are simply not a sound enough basis for rejecting contributions that might be* useful" [448]. An ethnography of software practice similarly noticed that little importance is ascribed to arguments based on empirical evidence [449]. *"The complexity people rarely dignify the pronouncements of the automatable-and-easy people by referring to them at all"* [184].

While engineering research should deal with problems that the researcher is uniquely qualified to attack, scientific research aims to gather data that are independent of the researcher's subjective interpretation. Stewart claims it would be excellent if questions like "*can an experienced practitioner duplicate this work from the text and references?*" "*became standard in computer science refereeing*" [463]. Such calls for improved empirical quality make it difficult to publish new ideas for attacking wicked problems. New ideas must first be thoroughly tested and compared to existing approaches, which is challenging because they often include new ways of framing the problem, incommensurable with previous work. Given a finite amount of resources (knowledge, time and money), an empirical study of an existing approach, with a hypothesis firmly rooted in previous research, will therefore easily achieve much higher scientific rigor than a project that has developed new ideas. Deciding funding for new research programmes based on publications and peer review of applications [526], further amplifies this trend. This motivates views that CS should replace "*publish or perish*" with "*demo or die*" [210].

Computer Science History and Experience

In 1976, Wegner [521] summarised the historical development of computer science research. He showed how *empirical* research into the nature of computing in the 50s, was replaced by *formal* analysis of computing concepts in the 60s, before the increased complexity of software lead to *engineering* research, developing tools and methodologies for practical management of this complexity. Average software costs were 5% of total system costs in the 40s, 70% in 1973, and estimated 90% by 2000 [521], indicating the growth of software's importance, size, and complexity. The speed of this development distinguishes computer science from other disciplines, and the co-existence of pre-scientific empirical research, formal science and engineering has increased the temperature of meta-scientific discussions.

Mathiassen [337] summarises the history of software development *practice* into three eras. Until the mid 70s, it was mainly a technical discipline, applying automation to improve productivity. The second era lasted until the late 80s, and emphasised effectiveness, integration and support. Analysis and design replaced programming and management as the core skills. In the third era global networks, collaboration, process orientation and knowledge management are key enablers, and domain knowledge becomes increasingly important. Variety, complexity, social dependence and multiplicity of skills have increased, and with them the need for interpretivist, practice-oriented research.

Tichy [490] and Ledgard [302] both make a point of the fact that object oriented programming never has been experimentally proven to improve productivity or quality of software, other than in case studies, which they disregard. They do not ask *why* these approaches have become popular with practitioners despite the lack of conclusive evidence in their favour. Wegner postulated in 1976 that while programming languages were critical to overall software development in the 50s and 60s, they may have become non-critical in the 70s, because further improvements would be of marginal importance for the whole software lifecycle [521]. Later development has supported this hypothesis [337]. Nevertheless, programming techniques are regarded as an important research area for empirical software engineering [490], perhaps because programming is more easily subjected to controlled experiments than problem framing or design? Since empirical studies have not yet been able to establish the best paradigm for programming, it seems premature to award them a monopoly on software engineering research in general. Even if it may in the future be possible to separate software design from implementation [151, 323], analysis and design will still require an engineering approach.

Voas [506] provides an interesting account of what can happen when empirical and scientific approaches take over a research area. He describes how the software quality community has moved from an enthusiastic arena for discussing ideas to becoming complacent with the state of practice, even though none of the major challenges have been conquered. At current conferences, very few new ideas are presented, and Voas blames a number of proposed silver bullets that did not work for this state. Among them are process improvement, formal methods, new languages and object orientation, metrics and measurement, software standards (especially process standards), testing, computer-aided software engineering, and total quality management. He advocates less over-selling, increased participation of practitioners in research and validation on real systems, not toy laboratory examples, to remedy these problems.

Both theoreticians and practitioners seem to blame the other camp for promoting breakthrough silver bullets [76]. Scientists point out that the proposed silver bullets have not been empirically validated as silver bullets [302]. Practitioners think that silver bullets could exist only if the work can be made rational [151], deterministic and empirically testable like scientists claim. Another study of software quality management concluded: "*Although the people we spoke to appeared to be well acquainted with the quality management literature, and accepted the principles in theory, our analysis of the data indicates that these principles have not been taken up in practice. However it is also possible that theory does not reflect practice, and the software quality literature does not adequately attend to the practical issues people face*" [449].

An assessment of usability research in human-computer interaction (HCI) makes similar observations. Noting that he has been astounded at the amount of bad and irrelevant research, Olsen [376] requests more practical case studies and less laboratory research. He blames a lack of holistic thinking, e.g.

- ignoring that "*a change here almost always affects something elsewhere*" [376],
- becoming "*hypnotised by the fetish of efficiency, unwilling to consider competing factors*" [376],
- "*you don't want to listen to closely to your customers*" [376] because they often confuse symptoms with underlying needs.

He refers to an analysis that found the most usable web sites to be produced by experienced designers without formal usability training and without user-centred techniques as recognised by the HCI research community. On the other hand, companies that chose to employ a separate usability specialist, involved in analysis and testing but not design, did not produce usable sites. Olsen thus criticise the unwillingness of usability researchers to look beyond the boundaries of their own community, e.g. to recognise that requirement specification techniques are user-centred even though they do not use the same terms as usability specialists.

An ethnography of software practice [449] note the frequent use of the term '*paradigm*' as an indication of rapid pace of apparent revolutionary change. This observation points to other fundamental differences between evolutionary science, and engineering with frequent revolutionary paradigm shifts [293]. Schön's 'problem frames'

[445] might be a more appropriate term in the engineering context, but though 'paradigm' is often misused, it reflects a lack of established scientific frameworks for software development. Without such a framework, evolutionary science seems premature.

Software Engineering as Reflective Practice

Various new software development methodologies have been coordinated under the umbrella of *agile methods* [219]. Following the approach of reflective practice [445], these methods highlight the interaction of analysis and design rather than the establishment of stable requirements prior to design, thus emphasising [219]

- Individuals and interaction over processes and tools,
- Working software over comprehensive documentation,
- Customer collaboration over contract negotiation, and
- Responding to change over following a plan.

Reflective systems development (RSD) [337] explicitly acknowledges its foundation in Schön's work. The approach shares roots with participatory design (PD), emphasising the role of users as active participants in software development. However, unlike PD, RSD also deals with the practice of software engineers.

What Makes Software Different?

A number of features makes software less suited for scientific enquiry than other branches of engineering:

- *Human* users influence information systems more than other engineering products. In technical terms, humans are "*variable and highly non-linear components*" [108]. Positivist scientific methods can test information systems as purely technical artefacts, verifying whether the system conforms to its specification. Interpretive approaches are needed to understand complex and evolving *sociotechnical systems*, validating to what extent an IS meets real user needs.
- Software is not controlled by *laws of nature* [210] which can be used for deriving principles that determine "*the boundary of what is safely possible*" [514].
- Software is *tangible mathematics*, uniquely suited for rapid, incremental build-and-test approaches. This explains why mathematics is so unimportant to the practitioner, and why the engineering approach works so well [514].
- Software is not *manufacturing*. New units of software have negligible cost, "*software is pure design*" [151]. Portraits of engineering as rational, formal, and science-driven [4, 114, 151, 323] does not fit software development. While industrial engineers rationalise production processes, software engineers shape the processes of use. This outsider perspective, sometimes articulated as a dichotomy between craft and engineering or even engineering and design, is thus not shared by software engineering professionals [306, 449]. Experience has shown that methodologies based on precisely defined processes and quantitative tools, is far less effective for design than for manufacturing [151].
- Software enjoys a more rapid pace of innovation than most other industries [114], and time-to-market is of greater concern [219].
- Information systems based on scientific theories like language action [124, 469, 536] and issue-based argumentation [113, 190, 452] have failed in practice. Reality was more complex then these theories derived from empirical analysis could account for. The step from scientific analysis to practical design is thus problematic.

- Many portray software as the most complex artefacts ever designed [353]. "*Indeed, the more complex the problem and its analysis, the more readily we can fall into the trap of focusing so intently and narrowly on the results of the analysis that we forget the fundamental assumptions upon which the analysis was based*" [398]. Formal science proofs are just the logical consequences of their assumptions.

Glass' study of videotapes of system analysts [184] is a rare scientific attempt to understand the nature of software engineering. Triggered by a chance encounter, he observed that 80% of the time, the analysts were just sitting there, doing nothing. These long periods were interrupted by short moments where the subjects wrote down what they had been *thinking*. Creative activities thus seem to take up most of the time. However, since it was impossible to agree on a measurable definition of creativity, more in-depth scientific studies could not be pursued. Hence, the most important aspects of software development do not seem to be empirical testable.

A.2.4 Information Systems Research

IS research deals with both development (engineering) and use. Some see IS as a social science, not an engineering discipline [3, 38, 359]. This perspective gives rise to research similar to organisational science. Both positivist and interpretive approaches have been applied in the study of information systems [273]. Interpretive perspectives assume that knowledge about reality is gained through *social constructions* such as language, shared meanings, documents, tools and artefacts. They focus on the *complexity* of human sense-making, attempt to understand phenomena through the meanings that people assign to them, and aim to understand the *context* of an IS and the processes whereby the system influences and is influenced by its context [273].

IS engineering is about the design of systems for information processing in an organisational context. Hence, a grounding of requirements in organisational, social and human sciences is paramount. Fekete [166] discusses what kind of contributions IS research can make. Categories of contributions include new (better) ways to provide known functionality, new (better) functionality, explanation of a phenomenon, theorems, general techniques, and reflective historical accounts of a project. Most of these categories are examples of *repertoire building* [445], though historical accounts may help research on the process of reflection. Explanations and general techniques can include new paradigms for framing problems.

A.2.5 CSCW - Computer Supported Cooperative Work

CSCW is the inter-disciplinary study of how people use information technology to communicate, coordinate and collaborate in groups [203]. CSCW occupies the middle ground between single-user-computer-interaction (HCI) and organisational IS. CSCW has been influenced from both sides, but a number of specialised concepts and research themes have also emerged for CSCW [414]. Ethnographic workplace studies [407, 448] have enjoyed a special standing among social science approaches to CSCW. Portrayed as naturalistic inquiry, refusing any ontological commitments [407], ethnomethodology takes an anti-theoretical starting point, arguing that the scientists should avoid using theories as lenses for their perception, that theorising should come after the study. Laboratory experiments cannot capture the prolonged, situated and contingent nature of social activity. Theoretically guided workplace studies ignore important aspects not pertaining to the hypotheses to be tested, and thus can not as easily inform design [407].

These views have met criticism from competing disciplines, claiming e.g. that the openness to impressions due to lack of hypotheses is illusionary, as well as the claim to be void of strategic opportunity [448]. Hybrid approaches that combine ethnomethodology with more theoretically oriented methods are thus advocated, on the basis that the relationship between practice and theory will change as science progresses.

The status of experimental methods in CSCW has also been assessed [277]. Here wicked aspects were identified, e.g. that "*cultural and organisational issues affect the ability to evaluate cause and effect*" [277]. CSCW thus still need interdisciplinary and design-oriented research. Dewan [134] provides a promising approach, combining design and evaluation of collaborative systems. The research plan he proposes is not based on scientific methods, but on engineering, combining applications and infrastructure research. Dewan also shows that infrastructure research derive its problems from preceding engineering work, indicating the role of engineering in providing new practical problems for science.

A.3 Conclusions

In an engineering project the main objective is to solve wicked real world problems. Engineering research provides new design patterns for the discipline's repertoire of action, and new understanding of the reflective design dialogue with problem situations. By performing system development we gain different insights than by observing the development practices of others. A study that integrates organisational requirements, design, implementation, and usage surveys, may not reach the same level of scientific rigour as more narrowly focused work, but the potential practical utility should increase. The wide scope means that there may be parts of this thesis that appear naïve to experts of particular domains, and other parts that are not explored scientifically to their full conclusion. Engineering research will be more open-ended than science directed at incremental contributions to a specific sub-discipline. Ideas may be useful even though they are not proven or fully empirically validated.

The research methodology, problem framing and solution proposals of this thesis are all based on a view of knowledge intensive work as reflective practice [445] in communities of practitioners [527] where shared understanding, identity, meaning and repertoires of action are socially constructed. The core concepts of this thesis include models, interaction, openness, process, incompleteness, evolution, holism, emergence, reflection, multiplicity, learning, and social construction. These concepts help us to select research methodologies, frame organisational needs, and design information systems. The three levels are thus consistent, coherent, and form a harmonious whole. This work thus exemplifies the notion that problem framing and problem solving are inseparably intertwined [445]. There is of course a danger that this harmonious whole becomes a self-guarding, self-fulfilling system of beliefs. This problem is met by ongoing interaction with real world situations, and the application of the solution to problems other than the ones it was originally designed for. Generalisation of design principles and recognition of similar approaches in other domains should also help us to avoid closure.

Appendix B
Implementation Details

B.1 Architecture

The architecture of WORKWARE is described at three levels:

- Logical architecture of main components (Chapter 7),
- Implementation architecture with packages of code, and
- Runtime architecture with servers, clients, and related software.

The two latter are described below.

B.1.1 Implementation Architecture

The WORKWARE implementation is divided into 4 layers of Java code, as pictured to the left in Figure 85.

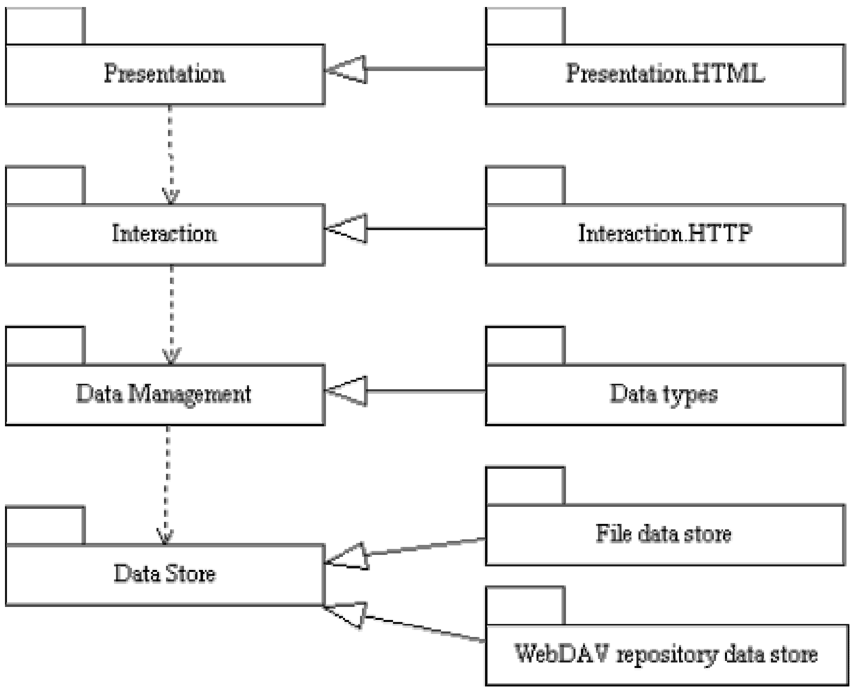


Figure 85. WORKWARE implementation architecture.

Each layer has its own Java package.

- The *Presentation* layer defines elements of the user interface. The presentation package contains general classes for user interfaces and customisation.
- The *Interaction* layer interprets requests from the user and decides how to answer them.

- *Data Management* handles all model data at runtime. The generic package contains implementations of data objects and attribute properties, as well as generic structures like lists, enumerations, extensional and intensional classes.
- The *Data Store* contains classes for persistent model storage.

The figure also shows extension packages that commit the general techniques to particular technological decisions, i.e. HTML user interface objects, client-server interaction across HTTP, the specific types of the WORKWARE PML, and two different persistent storage schemes.

B.1.2 Deployment Architecture

At runtime the WORKWARE code is running inside a Java Servlet container connected to a web server. The Java code builds complete HTML user interfaces dynamically according to general, organisational and user preference policies. These interfaces are then returned to the web client, via the web server, as an HTML document, using HTTP. This runtime architecture is depicted in Figure 86, which also shows the open nature of the system. Web pages and applications (e.g. corporate portals) can link directly to specific interfaces in WORKWARE, and WORKWARE can include links to other pages as services or information objects in the work process models. In this manner, WORKWARE may itself form the basis of a simple, process-oriented portal environment.

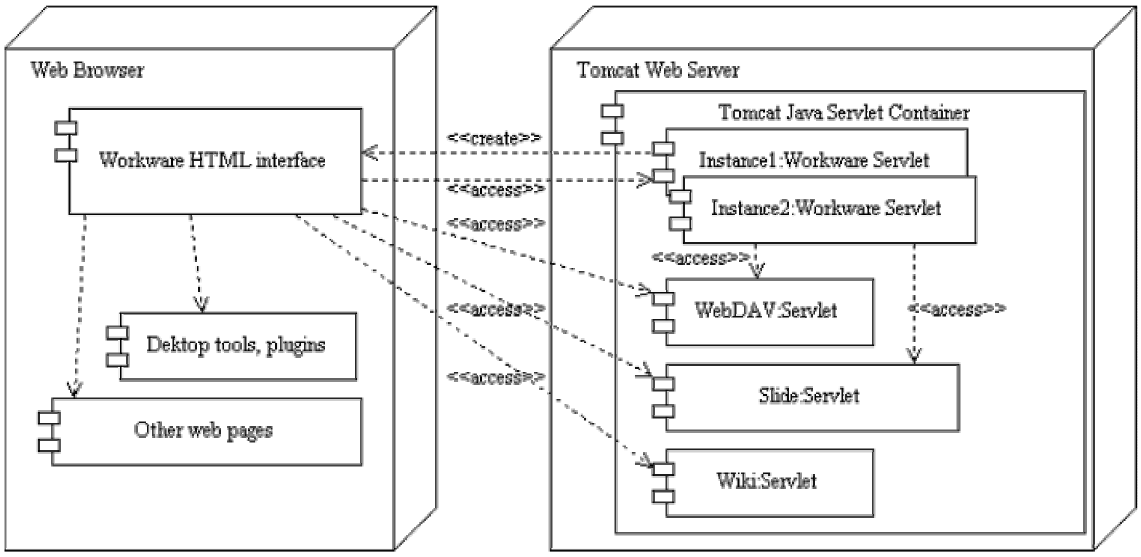


Figure 86. WORKWARE deployment architecture.

Each web server may contain several independent WORKWARE instances, e.g. for different projects or departments. These servers work on different sets of objects, and may have different services, configurations and language constructs. Over the lifetime of the prototype, we have used different operating systems on the server (WINDOWS 95, NT4, 2000, and ME), and a wide range of web servers (JAVA SERVLETRUNNER, MICROSOFT INTERNET INFORMATION SERVER, LIVESOFTWARE JRUN 2.2, later ALLAIRE JRUN 3.2, and finally APACHE TOMCAT 4.0). The basic architecture has handled this evolution well, though some migration problems have occurred (mainly to do with encoding of URLs and HTTP parameters). The open source TOMCAT web server currently in use offers standard WebDAV functionality, and has thus also been used as a lightweight model repository for the EXTERNAL infrastructure. We have also installed a WIKI servlet, which allows users to dynamically edit web pages, to add and remove pages. WIKI functionality is included as services in WORKWARE. Together, the two systems provide a rich set of functionality for project portals.

B.2 Implementation of the Language Metametamodel

The DataCatalogue is the access point to all modelled objects in the WORKWARE system. Each WORKWARE server instance has one DataCatalogue. In addition to the DataCatalogue, the main implementation class of WORKWARE's data management layer is the DataObject. This object implements interfaces for access these features of an object:

- Identifier,
- Name,
- Type (referring to a basic class),
- Java class that implements the object in the runtime system,
- Attributes (extensible set),
- Relationships to other objects, implemented as attributes whose values point to the other object,
- Help text that describes the object to the users.

All objects have a basic class that represents their primary classification. The basic class refers to a property of the object that is not expected to change. The modelling language constructs proposed in this thesis are implemented as basic classes. While the underlying data model separates basic classes from extensions, intensions, and enumerations, this is not visible in the ordinary user interface, where all are shown and manipulated in the same way (as *classes*). Every class has a membership (set of objects) and a template used for creating new objects.

Attributes are defined locally for each object instance, and stored in a Hashtable. Attributes have a definition that contains name (unique within each object) and expected value type (name of a class). *ListDataObjects* are used for multi-valued attributes. Each list has a default element type and an explicit set of elements. Lists are DataObjects, and may thus have additional attributes. More details on the use of lists are presented below.

B.2.1 Metamodelling Experience

The implementation architecture of WORKWARE (Figure 85) includes a package for specific data types. Originally we expected that the primitives of the modelling language would require specific functionality implemented in separate Java classes. Experience showed that such implementations were indeed needed for a few concepts (*WorkItem*, *Decision*, *Flow*), mainly for managing consistency of the workflow models within strict performance constraints. For most of the originally designed types, however, it became evident that the generic implementation classes were sufficient. Most of the functionality is anyway implemented as model interactors separate from the data management layer. The current basic installation of WORKWARE includes 100 modelling classes implemented by 5 generic and 8 specific Java classes. The mechanisms for reflection and metamodelling have thus greatly simplified the implementation itself.

B.2.2 Persistent Data Storage

The *Data Store* layer implements persistent storage of data objects (and classes). By default, each object is stored in a separate XML file on the WORKWARE server. When the server is started, all objects are read from file, and the system works on *DataObjects* in memory during its operation. When changes are made to the objects, updated files are written to disk. This scheme was chosen because it was the simplest to implement for a prototype. The interfaces are however generic and enables other storage mechanisms (e.g. databases) to be utilised in a future, more scalable version. For the EXTERNAL infrastructure, *DataObjects* are stored together in a model file with a different XML format. The model files can be stored locally or on a repository web-server.

B.2.3 Design Patterns for Flexible Data Management

Design patterns [173] provide a framework for assessing the quality of the WORKWARE design. In particular with respect to flexible data management, a number of design patterns have been proposed. By following these patterns WORKWARE thus conforms to state-of-the-art in software design. For object creation, Gamma et al. [173] defines these patterns:

- *Prototype* creates new objects by copying prototype instances rather than instantiating a class. This pattern is implemented as templates in WORKWARE. The *DataCatalogue* works as a *prototype manager*. Prototypes allow classes to be specified at run-time, and simplify the class hierarchy.
- *Abstract factory* is an interface for creating families of related or dependent objects without specifying their concrete classes. In WORKWARE the abstract interface *DataStore* is implemented by two concrete classes *FileDataStore* and *WebDataStore* that create objects at system start-up based on different persistent storage formats.
- *Factory methods* are abstract interfaces for creating objects that allow subclasses to decide which class to instantiate. *DataCatalogue* implements a *parameterised* factory method for creating new objects based on a parameterised class name. This method is used by the *DataStore* when creating new instances at start-up, and as a first step in the template cloning method for *DataObjects*.
- *Builders* separate the construction of a composite object from its representation, so the same construction process can create different representations. This scheme is utilised in WORKWARE for creating the structure of properties (including lists for multi-valued attributes) for each new object. Since each object may have a unique property structure, the construction process cannot be hard-coded.
- *Singleton* provides a global point of access, and is conventionally implemented as a class that has only one instance. Since multiple WORKWARE server instances can run inside the same namespace, each has its own *Context* object, which provides references to other singletons, such as the *DataCatalogue*, for the server in question.
- *Composites* implement tree structures, letting clients treat individual objects and compositions uniformly. In WORKWARE, this is exemplified by extensional super-classes also being classes.
- *Bridges* de-couple an abstraction from its implementation so that the two can vary independently. Parsons and Wand have argued object classes in analysis and domain modelling should be representational, not implementational [395]. In WORKWARE, this is achieved through bridges between user level and Java implementation classes.

- *Iterators* are implemented by *ListDataObject* in WORKWARE. These lists are used whenever access to a set of components is needed, whether the components are relationships, attributes, objects, or classes.

In addition to these patterns for data management, WORKWARE also utilises other patterns in other layers of its architecture.

B.3 Classes in a Default WORKWARE Installation

This section describes the full metamodel that is installed alongside the program code. Some of these classes are mandatory for the system to work properly. We briefly describe each class, how it is implemented and its most important relationships. Since no general graphical notation has been defined yet for metamodelling in WORKWARE, and UML is poorly suited, we just provide a textual description. Class names are printed in **bold** (or in a heading) when they are first introduced, and in *italic* when they are referred to later.

B.3.1 Data Management

The classes below are all members of the *Data Management* extension. They may be regarded as metaclasses.

Object

All objects are members of this system-defined class. Objects are identified by their unique *ID*, and contain *Name*, *Type*, *Implementation* (name of Java class) and *Help* text attributes.

Class

All classes are members of this class. They are identified by their *name*, so in individual objects, the value of an attribute with type *Class* is just a *Text*.

Attribute

Attributes are identified by their *name*, just like classes. This metaclass is typically used in a context where a specific object or class is identified by other attributes. It allows customisation at the attribute level. An attribute instance also defines *value* and *type*.

List

A list contains a number of *Elements*. The *Element type* attribute points to a *Class*.

Extension

An extension is a list whose elements are *Classes*. (The *Element type* attribute's value equals *Class*). It also identifies the default subclass that is used for instantiation. The *Extension* template is used for creating new *Extension* classes. New subclasses may be added at any time, just like new elements are added to any other list.

Intension

An intension identifies one or more (through a list) classes that it specialises. An attribute called *Query* defines the criteria that members must fulfil. The criteria compare attributes to values defined statically as part of the *Intension* object. A template is used for creating new *Intension* classes.

Simple Data

This *Extension* groups all the classes whose values are not objects, but values, i.e:

- *Character*
- *Number (integer)*
- *Number (real)*
- *Date*
- *Text*
- *URL*
- *Yes or no (boolean)*

Attributes with these types all have data values stored directly in attribute definitions.

Enumeration

Enumerations are *lists* that have *simple data* (any subclass) as their *element type*. The *enumeration* template is used for creating new *enumeration* classes. The element set may be altered at any time, just like new elements are added to an ordinary list.

UserData and SystemData

These *extension*s separate classes that are used by solution providers from those used by end users as part of normal operation.

B.3.2 Process Modelling

The classes used for process modelling are all part of this *extension*.

WorkItem

The *WorkItem* template contains a range of attributes, such as *Status*, *Description, Due date*, *Parent*, *Responsible*, *Customer*, *Services*, *Documents* etc.

Project

A project has a *Name*, a *Model* (filename) and a *Root workitem*.

Flow

A flow connects two connectors (*From* and *To*). During enactment, the boolean *Activated* attribute defines its state.

Decision Connector

A decision connector has a *list* of *Input flows*, a *list* of *Output flows*, a reference to its *Owner* workitem, and a logical operation that relates numerous inputs or outputs. The *Connector type* attribute defines whether the relation applies to multiple inputs or outputs. Like flows, connector states are defined by the *Activated* attribute.

LogicalOperator

This *enumeration* has three values, "AND", "XOR" and "" (unspecified).

ConnectorType

This *enumeration* has two values "Join" and "Fork".

Connector - Join

This *intension* contains all connectors whose *Connector type* equals "Join".

Connector - Fork

This *intension* contains all connectors whose *Connector type* equals "Fork".

Resource

This *extension* groups all objects that may appear as resources on a workitem, i.e. *Person*, *Document* and *Service* (for tools).

Person

A person is both a potential role-filler on workitems and a potential user of the system. A number of attributes are associated, e.g. username, full name, password, email address, phone number, web page etc.

Document

Documents contain a logical name and (if it exists) the *URL* of the real document.

Status

This *enumeration* contains all the statuses that a workitem may have, i.e. *Planned*, *Waiting*, *Ready*, *Ongoing*, *Suspended*, *Finished*, *Terminated*.

Document Status

This *enumeration* may be used to classify documents based on their status.

B.3.3 Service

This *extension* contains all objects that represent pieces of functionality in the system. It is itself a subclass of *Resource*, and indirectly of *Process Modelling*. It has three basic subclasses.

Worklet

A worklet is a piece of WORKWARE's internal functionality. Typically, it invokes one of the system-defined *operations* in a parameterised manner. Parameter values may be extracted from the request (the data contained in the current user interface).

URLService

A URLService is a piece of external functionality, which is performed by opening up a URL. Desktop tools are also modelled as URLServices. They are opened by returning an empty document with the appropriate MIME type.

Menu

A menu object is an internal node in the WORKWARE Explorer menu structure. It contains a submenu as a *list* of *services*. The menu may however also contain other objects, e.g. *Projects*, *ItemLists* and *WorkItems*.

B.3.4 Service Management

A number of classes are used for controlling which services are made available in each user interface. Two *extensions*, **Global Service Management** and **User Service Management** group the concrete classes.

ServiceConfiguration

ServiceConfigurations contains a list of *Included services* and another list of *Excluded services*. They may also contain other attributes that define the scope of the configuration. Any configuration may thus be overridden by more local configurations, through cancellation inheritance.

ClassServiceConfiguration

These objects define the *services* that should apply to all objects of a specific *class*.

UserClassServiceConfiguration

These objects add or remove *services* for particular *persons*. For instance, novice users do not see the services "Add attribute" or "Remove attribute" on *WorkItems*.

ObjectServiceConfiguration

These objects define the *services* that should apply to a specific *object*, e.g. tools needed to perform a *workitem*.

OperationServiceConfiguration

These objects define which *services* should be available in different modes of *operation*. The service "Save" is for instance available in *EditObject* mode, but not in *ViewObject*.

UserOperationServiceConfiguration

Some *persons* may have access to extra services, such as *system administration*.

B.3.5 User Interface Preferences

User interfaces are controlled by policy objects. They are grouped into three subclasses (extensions): *User Interface Components*, *Global Preferences* and *User Preferences*. All user interface components are members of the basic class **GUIComponentClass**. Experience has shown that this class should have been specialised, e.g. to separate forms from images and fields. *GUIComponentClass* objects contain a reference to the Java class that implements this user interface component or container (typically an HTML element), and possibly some parameters for customisation (e.g. size, font, colour) stored as ordinary attributes.

Global Preferences

These policies define the default look, feel and content of the user interface. Three different subclasses (*extensions*) group the policies into different levels (the whole form, each object, and each attribute).

Form Preferences

These policies define which container *GUIComponentClass* is used for displaying objects of a certain class in a given *operation* mode. These concrete subclasses exist:

- *ObjectEditStyle*
- *ObjectViewStyle*
- *ObjectSearchStyle*

List Preferences

List preferences control the default content and appearance of list interfaces. **ListEditStyle** and **ListViewStyle** define which containers (*GUIComponentClass*) are used for displaying the whole list of objects, while **AttributeList** contains the attributes (columns) to display for each object. The appearance of each attribute is controlled by attributeviewstyles (see below). The set of objects to show may be given as result of a *Search* operation, but it may also be defined as an *ItemList*.

ItemList

ItemLists are objects that define a dynamic grouping of other *objects*. A *Type* attribute defines the class of objects to search among, and *Query* defines the criteria that objects must fulfil. In addition, objects may be explicitly added to or removed from the list, by placing them in the *Included items* or *Excluded items* lists. *Itemlists* are used to define worklist, where users may add or remove workitems. Such *lists* may also have extra attributes, such as *Owner* (a *Person*), which defines their scope.

Attribute Preferences

Attribute preferences define what user interface component is used for displaying the value of each attribute. **AttributeEditStyle** and **AttributeViewStyle** define general policies for all values that have a given class, while **SpecialAttributeEditStyle** and **SpecialAttributeViewStyle** define more local policies for individual attributes of a given class, such as "*WorkItem.Status*".

User Preferences

Personal preferences may be defined for any of the above mentioned general user interface policies:

- User Form Preferences (extension)
  - UserObjectEditStyle
  - UserObjectViewStyle
  - UserObjectSearchStyle
- User List Preferences (extension)
  - UserListEditStyle
  - UserListViewStyle
- User Attribute Preferences (extension)
  - UserAttributeEditStyle

- UserAttributeViewStyle
- UserSpecialAttributeEditStyle
- UserSpecialAttributeViewStyle

There is also an additional class, **UserMainContainerStyle**, which allows a user to select among alternative components for the main interface, e.g. between multi-frame and single-frame containers.

B.3.6 Awareness

The awareness notification and event logging services are customised by a number of classes, grouped under the common extension *Awareness*. The simple value class **Event** defines which *persons* caused which *operation* (read, write, update etc.), on which parts (*attributes*) of which *object*, at which *time*. Classes that log events (e.g. *WorkItem*) are members of the extension **ObjectsWithEventLog**.

Awareness Profiles

A basic **AwarenessProfile** contains two lists of event filtering *Lenses*, *Included lenses* and *Excluded lenses*. A Lens belongs to one of the following classes:

- *EventListLens*, removing all events explicitly listed,
- *EventTypeLens*, removing all events of a given type, e.g. all "*Read*" events,
- *EventUserLens*, removing all events caused by a given user, e.g. myself,
- *TimeLens,* removing all events that happened before a given point in time,
- *RelativeTimeLens*, removing all event older than a given duration.

Each user may have a personalised **UserAwarenessProfile**, which is applied to all objects, and local **User2ArtifactAwarenessProfiles**, which apply to a single object (e.g. a *workitem*) only.

B.4 Implementation of Interactive Activation

The model interactors are the components that activate process models in WORKWARE's architecture. The core components of WORKWARE implement generic mechanisms for sharing model data among distributed users. This includes simple, customisable mechanisms for viewing, editing, listing and searching the modelled data. The core thus supports generic articulation and manual activation of models of any kind. This section gives a brief overview of the core functionality in the system, as most of the interactors reuse both the overall design as well as generic interfaces for data access, event notification, user interfaces and customisation.

B.4.1 Dynamic Generation of User Interfaces

Figure 87 shows the core components involved in generating and controlling the user interface of WORKWARE. When a parameterised request is received from the client (via a web browser and a server as shown in Figure 86), the parameters are inspected in order to identify the operation that the user wants performed. After the operation is performed (e.g. the changes supplied by the user are sent to the *Data Management* layer), the WORKWARE core servlet asks its *PresentationCatalogue* which user interface container should be used to reply to this request. This container typically is an HTML form or table. Which container the *PresentationCatalogue* selects, depends on the user interface profile for the current user. User interface profiles are ordinary model objects.

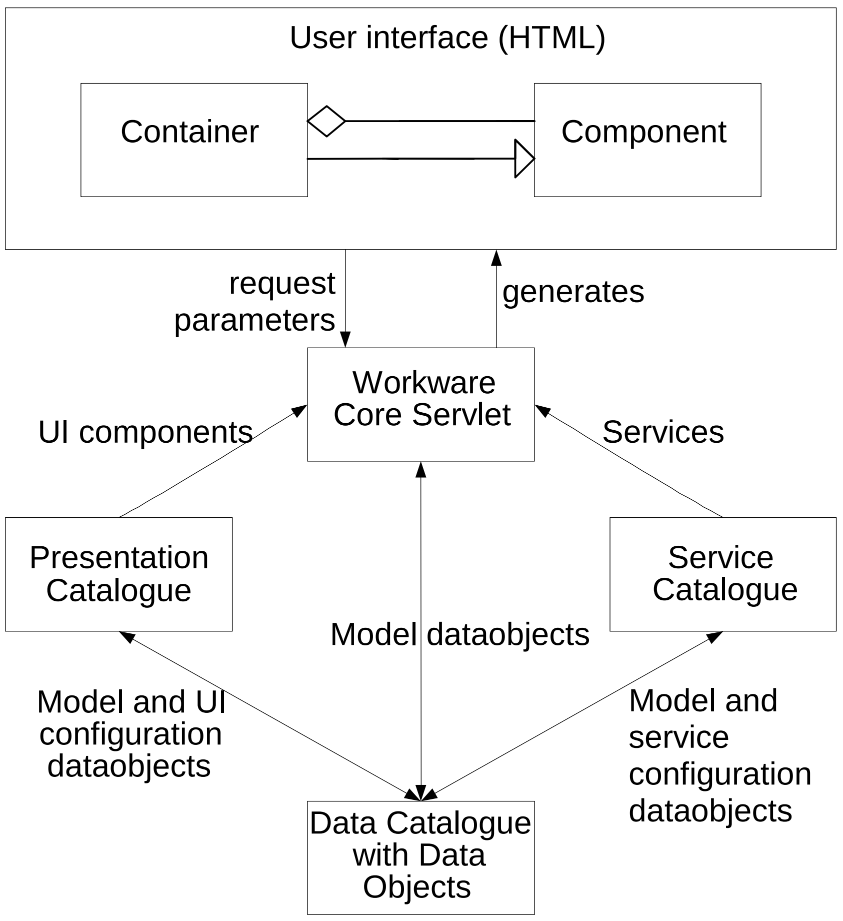


Figure 87. Core W*ORKWARE components for user interaction.*

The next step is for the core servlet to notify the container object about the detailed parameters and model data of the request. This is done through a set of standard interfaces that different component classes may implement:

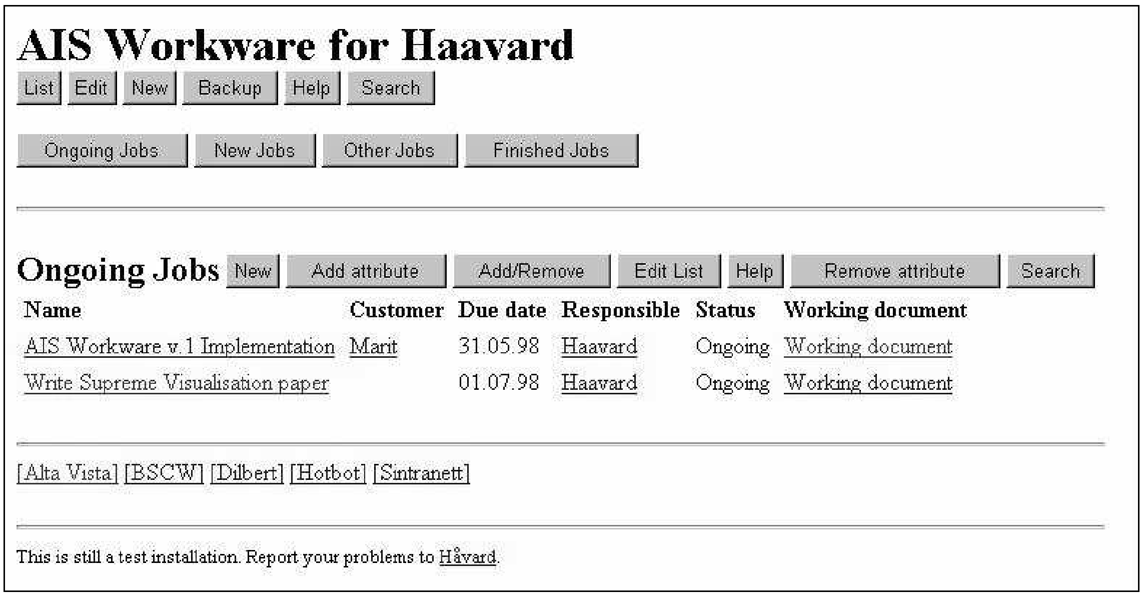
- *UserSpecificComponent*, adjusting its presentation to the current user (e.g. depending on access rights),
- *OperationComponent*, adjusting to the current operation. For instance the same container may be used both for viewing and editing some piece of information. Depending on the *operation* of the current request, different user interface components (for view or edit) will be selected for the various data to be presented.
- *ObjectListComponent*, for displaying a list of objects, thus it takes the list of objects to display, the attribute to sort by etc. as input.
- *ObjectComponent*, for working on a single object, taking the object in question as input parameter.
- *TypeComponent*, adjusting to the type (class) involved. The type is often derived from the object(s) involved, but in cases where the object belongs to a number of classes, different behaviour may be needed in different contexts, depending on which class the object currently is typecasted as.
- *Selector*, able to group the content and show only parts of it (the currently selected group) at a time. One example is the tabfolder used for worktops.

- *ContainerWithServices*, capable of including a customisable set of services that the user may invoke.
- *TailorableComponent*, a generic interface that allows more extensive customisation, e.g. according to user-defined parameters. *TailorableComponent* takes an ordinary model object (e.g. a profile) as input, and can be customised by any property of that object.

After the container has been parameterised through these interfaces, it is capable of returning the resulting HTML code. Typically it contains a number of components for different parts of the data involved, e.g. one component for each attribute of an object. The container ensures that the components are customised through the interfaces listed above. These components may themselves be containers, e.g. a *TableRow* for each object in a *Table* acting as an *ObjectListComponent*. Containers are themselves able to integrate all of these components. In the case of the HTML interface, this integration is achieved by concatenating the HTML code for each of the components, adding element separator tags (e.g. separating table rows) when needed. In addition to components for viewing or editing the data from interactive models, a user interface in WORKWARE includes components for invoking *services*. Which services should be included for each request, is dynamically controlled by the *ServiceCatalogue*, and subject to user-defined *ServiceConfiguration* models.

B.4.2 Technological Evolution

The available set of HTML containers and components has increased throughout the lifetime of WORKWARE. The general scheme for dynamic user interface generation and customisation presented above has proven capable of handling this evolution. Although specific interfaces like *Selector* has been added, the core customisation interfaces as well as the modelled policy objects that controls the user interface, have been relatively stable throughout the period. Figure 88-Figure 92 shows the evolution of the WORKWARE user interface.



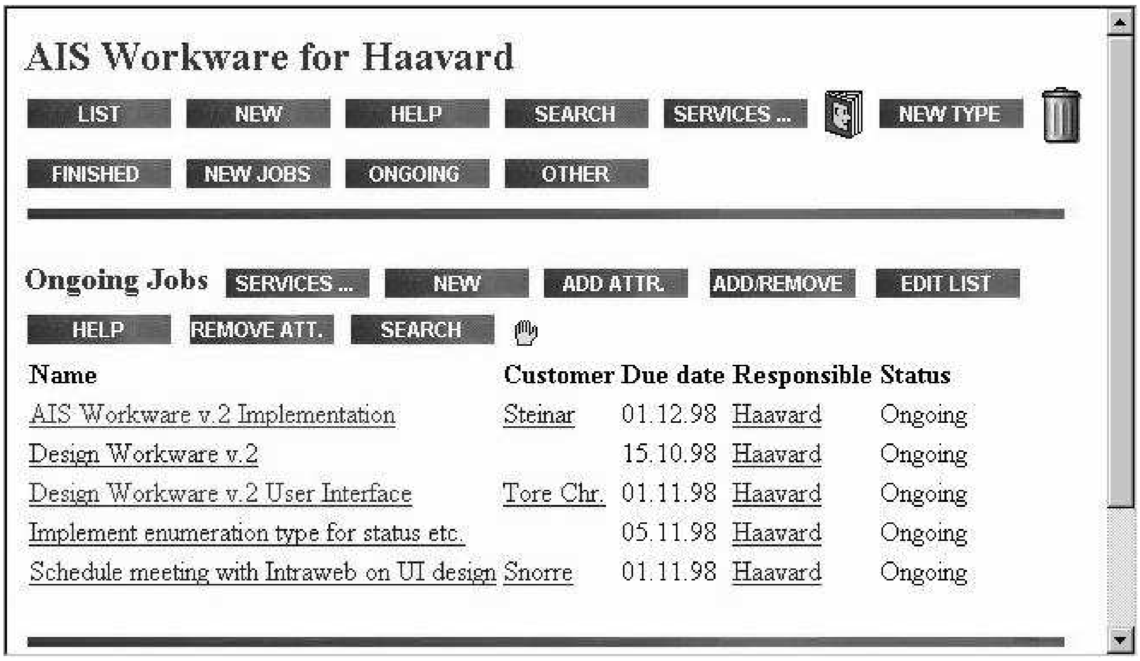
Name	Customer	Due date	Responsible	Status	Working document
AIS Workware v.1 Implementation	Marit	31.05.98	Haavard	Ongoing	Working document
Write Supreme Visualisation paper		01.07.98	Haavard	Ongoing	Working document

[Alta Vista] [BSCW] [Dilbert] [Hotbot] [Sintranett]

This is still a test installation. Report your problems to Håvard.

Figure 88. WORKWARE user interface, version 1, 1997.

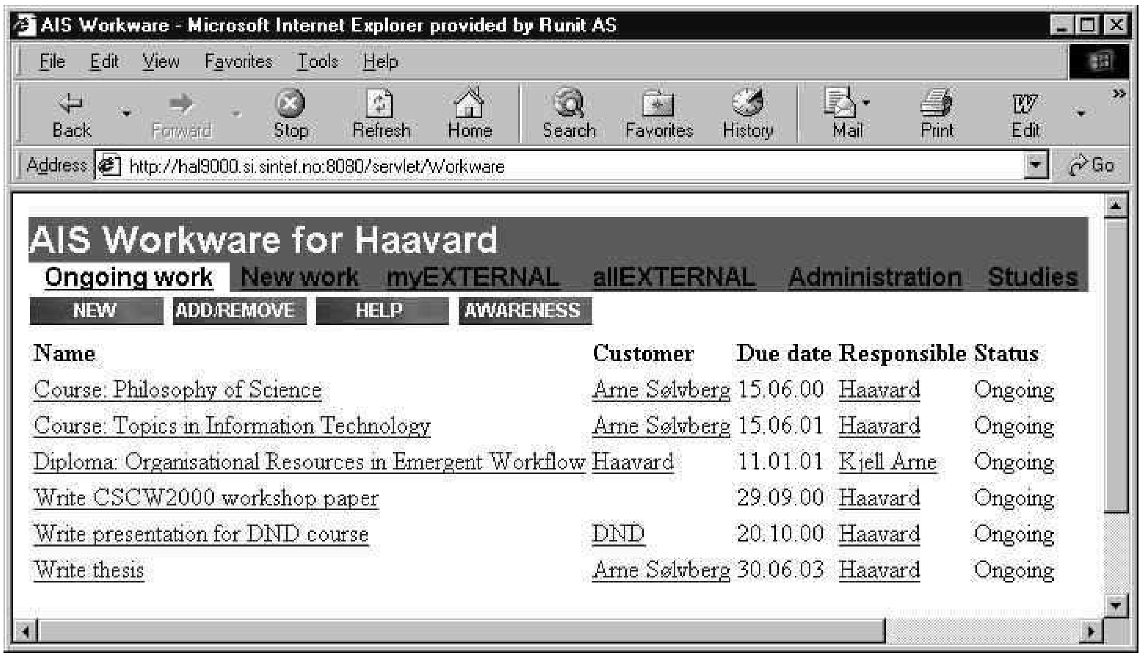
The first version included only basic HTML elements like Form, Anchor, Button, Text, and various Input fields. Version 2 (Figure 89) added images with anchors, greatly improving the look and feel of the system, without interfering much with the underlying container layouts.



Name	Customer	Due date	Responsible	Status
AIS Workware v.2 Implementation	Steinar	01.12.98	Haavard	Ongoing
Design Workware v.2		15.10.98	Haavard	Ongoing
Design Workware v.2 User Interface	Tore Chr.	01.11.98	Haavard	Ongoing
Implement enumeration type for status etc.		05.11.98	Haavard	Ongoing
Schedule meeting with Intraweb on UI design	Snorre	01.11.98	Haavard	Ongoing

Figure 89. WORKWARE user interface, version 2, 1998.

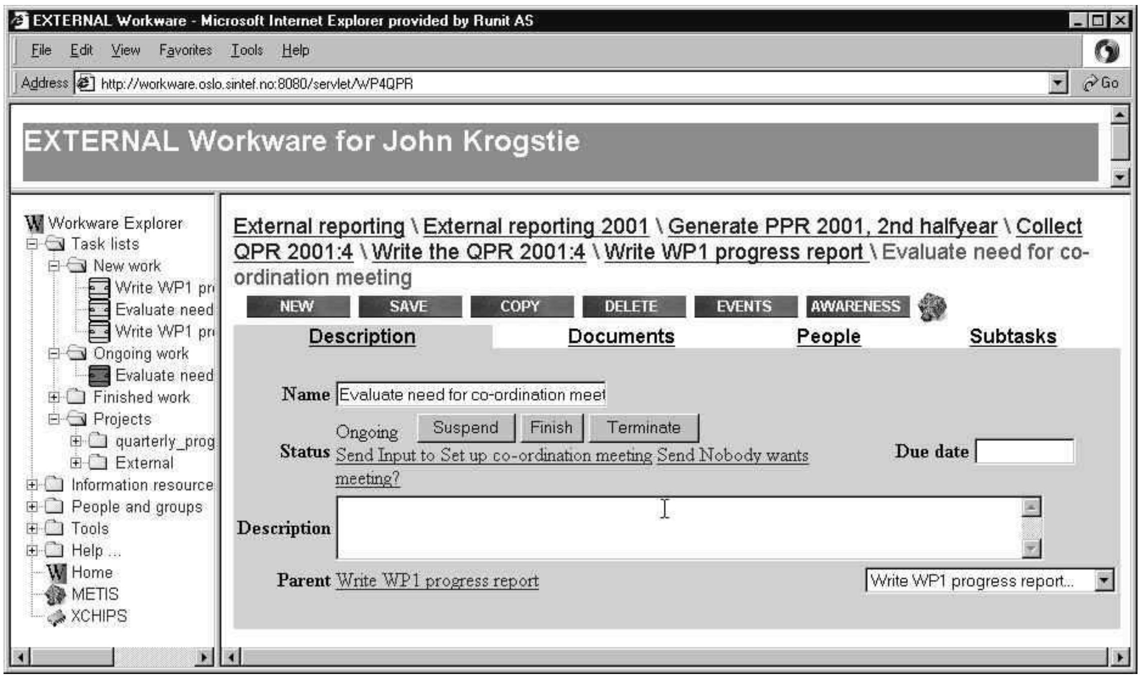
Version 3 (Figure 90) added the tabfolder interface (implemented as a table with differently coloured cells) because the core objects (WorkItems) as well as the menu structures were getting to complex to be shown in all together.



Name	Customer	Due date	Responsible	Status
Course: Philosophy of Science	Arne Sølvberg	15.06.00	Haavard	Ongoing
Course: Topics in Information Technology	Arne Sølvberg	15.06.01	Haavard	Ongoing
Diploma: Organisational Resources in Emergent Workflow	Haavard	11.01.01	Kjell Arne	Ongoing
Write CSCW2000 workshop paper		29.09.00	Haavard	Ongoing
Write presentation for DND course	DND	20.10.00	Haavard	Ongoing
Write thesis	Arne Sølvberg	30.06.03	Haavard	Ongoing

Figure 90. WORKWARE user interface, version 3, 1999.

The menu structure (consisting of hierarchically grouped services) was getting increasingly larger, so for version 4 (Figure 91) it was decided to move away from the single-frame interface and add a separate frame for the menu structure, called the WORKWARE Explorer. While this change was highly appreciated among users, it complicated the interface and made it less portable. However, all the old components are still available, so that e.g. for mobile devices, we are able to configure the system to use the older interface.


EXTERNAL Workware - Microsoft Internet Explorer provided by Runit AS

Address http://workware.oslo.sintef.no:8080/servlet/WP4QPR

EXTERNAL Workware for John Krogstie

External reporting \ External reporting 2001 \ Generate PPR 2001, 2nd halfyear \ Collect QPR 2001:4 \ Write the QPR 2001:4 \ Write WP1 progress report \ Evaluate need for co-ordination meeting

Name: Evaluate need for co-ordination meet

Status: Ongoing

Send Input to Set up co-ordination meeting Send Nobody wants meeting?

Due date:

Description:

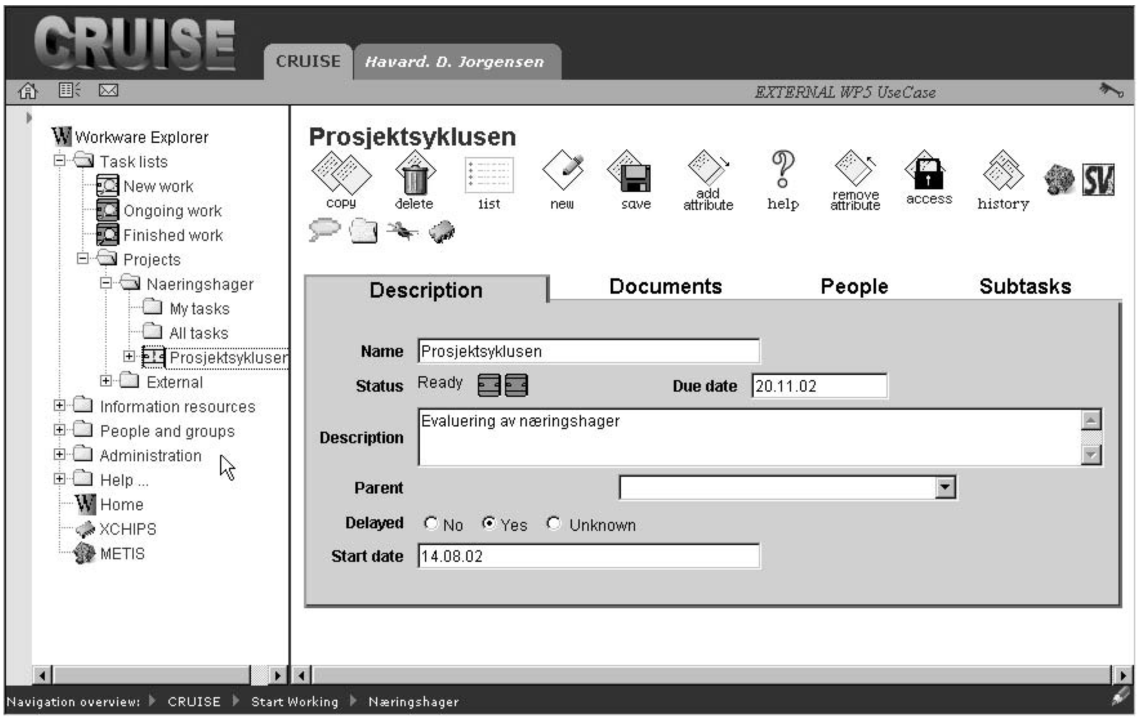
Parent: Write WP1 progress report

Figure 91. WORKWARE user interface, version 4, 2001.

Finally, version 5 introduced stylesheets, making the interface look more up to date and enabling user organisations to user their own colours, fonts, sizes etc. Figure 92 shows another key feature of the WORKWARE user interface, its capability to integrate with other web applications and HTML pages. Since all functionality of the system can be invoked as URLs and the whole user interface is HTML, it is simple to integrate it into, e.g. a corporate portal. In Figure 92 the header frame of WORKWARE is replaced by that of the UEPS portal [491]. The UEPS menu is included in a minimised frame to the left of the WORKWARE Explorer, so users easily can switch between them.

B.4.3 Work Management

The work management tool is more tightly integrated with the core components than the other model interactors. This reflects the nature of the services it provides, which allows users to share updated descriptions of their work and report on progress through updating the state property of the items. The component thus does not provide any automated services, and consequently it is the specialised user interface components that constitute the work management tool. As described previously, these components are the worktops for editing and viewing single workitems, and worklists providing overviews of items that satisfies user-defined search criteria. Each form can be customised further as described in the previous subsection. Users may for instance select different HTML components for interacting with the various properties of workitems, and include a customised (modelled) set of services for each item. For worklists, users may also decide which properties should be shown for the items.



Prosjektsyklusen

Name: Prosjektsyklusen
Status: Ready
Due date: 20.11.02
Description: Evaluering av næringshager
Parent:
Delayed: No [✓ Yes] Unknown
Start date: 14.08.02

Figure 92. WORKWARE user interface, version 5, 2002.

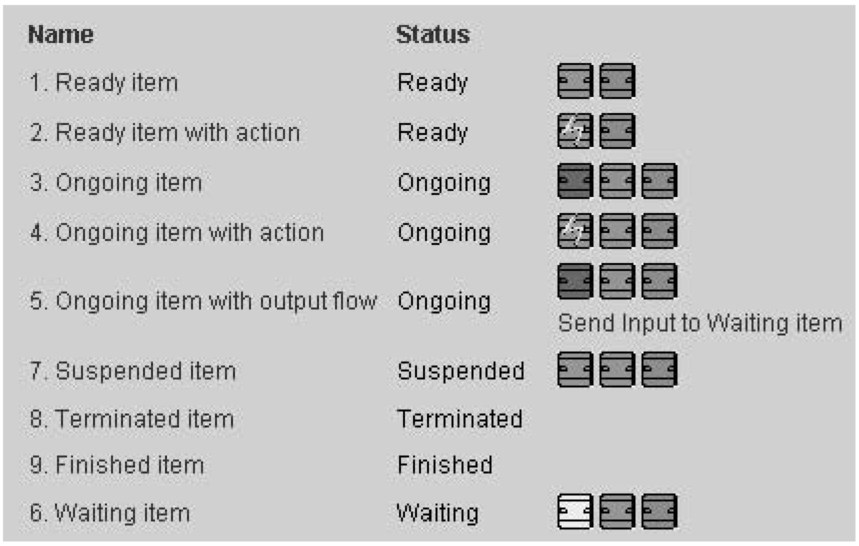
B.4.4 Interactive Enactment

The interactive enactment engine is mainly an automation tool, although it involves users in situating the interpretation of the model through a few specialised user interface components. The enactment engine contains these components:

- *EngineCore*, which subscribes to change events on all modelled workitems, decision connectors and flows. If the state or activated property of these objects is changed, the enactment engine sees to it that related objects are also updated according to the rules and state transition diagrams presented in Chapter 4. The engine may also add decisions to the worklists of responsible users in the case when reactive manual decision making is required.
- *EngineServlet* is responsible for replying to coordination services (state change) invoked by the users. It delegates the interpretation and activation of secondary effects to the *EngineCore*, and invokes the main *Workware* servlet to return the work management interface when it is done.
- *CoordinationServiceList* is a user interface component for editing the state properties of workitems. It displays the current state and includes other components for invoking the coordination services that applies to the current item in this state (buttons in Figure 91, images in Figure 92). Coordination services may alter the state of the item, or explicitly activate an outgoing flow. Which services are included is determined by the outgoing transitions from each state in the diagrams presented in Chap-

ter 4. This user interface component is used both in worktops and in worklists (so users can quickly report on state changes without having to open the full worktop).

- *FormForDecisionMaking*, a specialised interface for resolving ambiguously modelled decisions, e.g. because there are multiple inputs and/or outputs that can be activated at different times. The form includes a field for discussing the issues involved and components that lets the user activate output flows.



Name	Status	
1. Ready item	Ready	
2. Ready item with action	Ready	
3. Ongoing item	Ongoing	
4. Ongoing item with action	Ongoing	
5. Ongoing item with output flow	Ongoing	Send Input to Waiting item
7. Suspended item	Suspended	
8. Terminated item	Terminated	
9. Finished item	Finished	
6. Waiting item	Waiting	

Figure 93. Coordination services for workitems with different states.

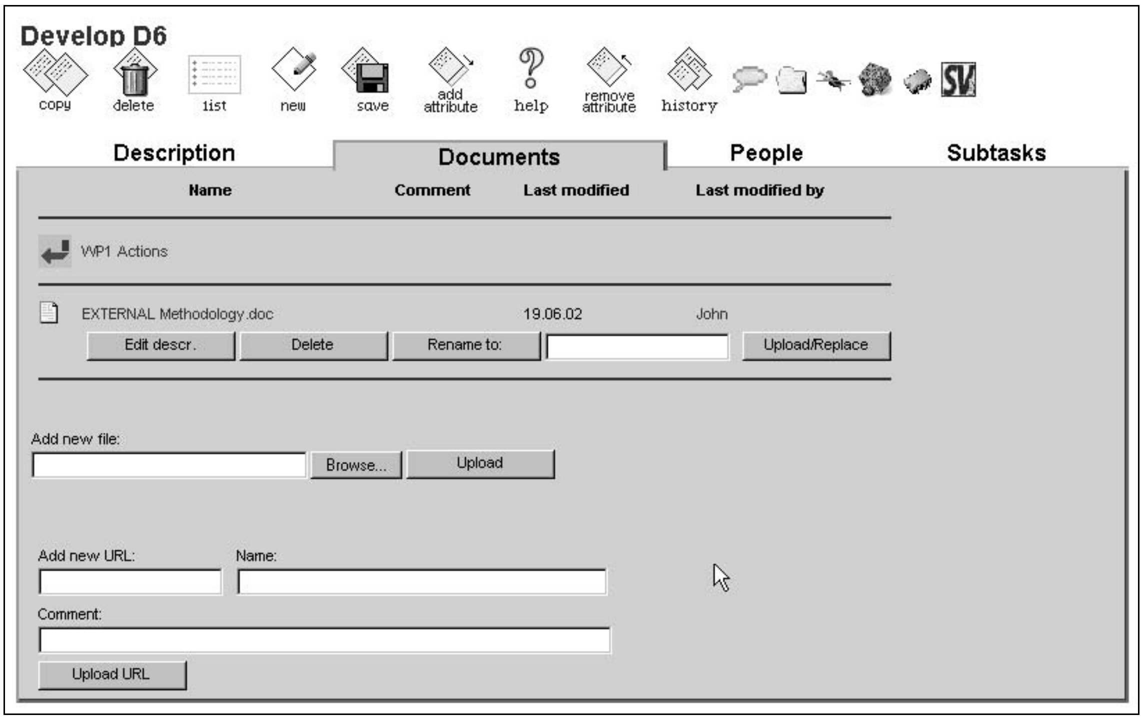
Figure 93 shows different instances of *CoordinationServiceList* for different workitems. The buttons next to the state have different colours depending on the state the item changes to when the service is invoked, and ToolTip help that explains the service. All items that are not finished or terminated have *Terminate* as their last service. *Ongoing* and *Suspended* items also include *Finish*, and services for moving between *ongoing* and *suspended*. *Ready* and *Waiting* items here allows *Start* (even though it in the case of waiting items corresponds to violating the modelled sequence of work). Item 2 and 4 have an invoked tool (*service*) attached to them, and thus have the tool service included (icon with lightning across the workitem symbol). Item 5 has an output flow that keeps item 6 from becoming ready. This flow can be activated from the *CoordinationServiceList*. The sorting order is by name for objects that are not ordered in sequence by flows (most of them), but item 6 is placed last because it is modelled in sequence after item 5. Note that the whole user interface is generated by work management, and that the *CoordinationServiceLists* are the only components from the enactment interactor. This exemplifies user interface integration of the interactors in the system.

B.4.5 Document Management

The basic principle for WORKWARE's document management interactor is to utilise process structures to classify and group documents, simplifying retrieval, management and contextual interpretation of the information. The component was originally designed as an example of ontology-driven information management in Intranets. In another example from in the same project, enterprise models were used to define Intranet navigation structures [365]. This design and information workspace metamodel is thus extensible to other kinds of interactive models. A simplified version was implemented in WORKWARE, using open source software. The interface between this component and the rest of the system is managed by a *WorkspaceManager* object (utilising the *façade* design pattern [173]). This object makes sure that changes to the document base are reflected in the process models. The component further includes servlets for uploading documents, defining links to existing documents (URLs), and the following user interface components:

- *DocumentList*, showing all the documents that satisfy current selection criteria, e.g. all document resources assigned to workitem,
- *UploadForm*, for selecting local documents and uploading them,
- *URLUploadForm*, for defining new document model objects based on existing documents somewhere in the World Wide Web.

Figure 94 shows how these components are included in the work management user interface of a workitem (Develop D6). First the document list shows all the documents on the item (*External Methodology.doc*), with services, then the upload forms for adding new documents to the task are shown below. The leftmost WORKWARE Explorer example in Figure 72 on page 169 shows that the flexible metametamodel of WORKWARE can support and utilise other forms of metadata on documents as well, in this case a specific document category is used for separating documents into submenus. In all, the document management component presented here is a simplified proof-of-concept prototype that has been used in some cases. It is not a full-fledged document management system, but such a system could be integrated into the framework. Lack of versioning, a cumbersome user interface, and access control, are among the limitations of the current implementation.



Name	Comment	Last modified	Last modified by
WP1 Actions			
EXTERNAL Methodology.doc		19.06.02	John

Add new file:

Add new URL:

Name:

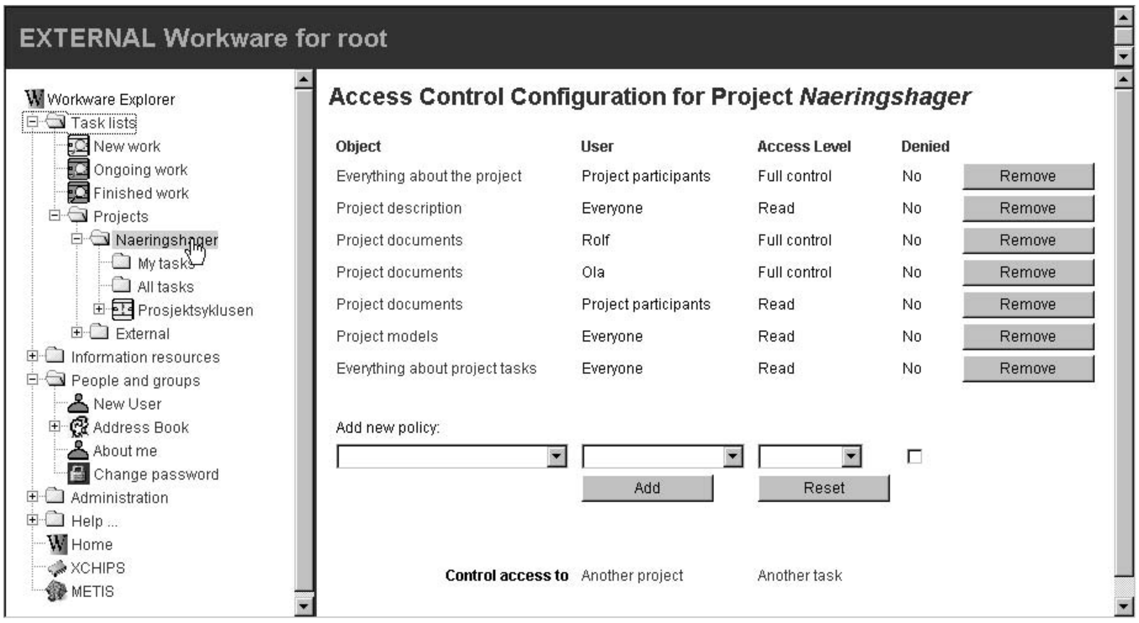
Comment:

Figure 94. Document management interface for the workitem "Develop D6".

B.4.6 Access Control

The overall concept and services of the model-driven access control component were presented in section 4.8. With respect to document management the implementation of access control remedies some of the shortcomming, in that an open source content management system (JAKARTA SLIDE [17]) is integrated. SLIDE supports distributed authoring and versioning according to the WebDav standard [235]. In addition to storage, retrieval and access control, locking, and versioning are also supported by SLIDE. These services were integrated into WORKWARE through these classes:

- *WorkwareACLUpdater*, a component that listens for changes to all the involved model objects (workitems, documents, persons, projects), and makes sure that these changes are reflected in the access control structures.
- *WorkwareACLServlet*, controlling user interaction in access control, and implementing the mapping between process structures and access control structures.
- *ACLForm*, a user interface for defining and viewing current access control policies, in a vocabulary consisting of process model terms rather than files and directories (wherever possible). Figure 95 shows an example access control form.
- *ExternalACLInterface*, encapsulating and providing a single point of access from WORKWARE to all the various SLIDE objects.



Access Control Configuration for Project *Naeringshager*

Object	User	Access Level	Denied	
Everything about the project	Project participants	Full control	No	Remove
Project description	Everyone	Read	No	Remove
Project documents	Rolf	Full control	No	Remove
Project documents	Ola	Full control	No	Remove
Project documents	Project participants	Read	No	Remove
Project models	Everyone	Read	No	Remove
Everything about project tasks	Everyone	Read	No	Remove

Add new policy:

Add Reset

Control access to Another project Another task

Figure 95. User interface for access control to a project.

The mapping between process structures and access control structures (files and directories) is central in this design. For instance, reuse of access rights by inheritance is supported down the directory hierarchy of SLIDE. Table 14 shows this mapping as it is currently implemented.

Model Object	**Slide Object**	**Slide Example**
Person	In Slide's directory for users	/users/Haavard
Project	Project model url (xml file),	/Projects/model1.xml
All project content	Inside directory whose name is the project-url without extension	/Projects/model1/
Document (full text)	Document url	/Projects/model1/Documents/ thisdocument.doc
Workitems	Unique id of workitem inside project model file, below project content directory	/Projects/model1/oid23
Persons assigned to project/workitem	Separate directories for each role type below the directory of the project or workitem, contains links to persons	/Projects/model1/oid23/ Responsible /Projects/model1/Participants
Information resources	The Documents directory below the directory of the workitem, contains links to full text documents	/Projects/model1/oid23/ Documents/thisdocument.doc
Properties	Directory named like the attribute, in the object's directory	/Projects/model1/oid23/Name

Table 14. Mapping between model objects and access control objects.

List of Figures

List of Tables

References

1. Abbot, K. R. and Sarin, S. K. *Experiences with Workflow Management: Issues for The Next Generation*, ACM CSCW Conference, 1994.
2. Abdel-Hamid, T. K. and Madnick, S. E. *Lessons Learned from Modeling the Dynamics of Software Development*, Communications of the ACM, vol. 32, no. 12, 1989.
3. Adam, F. and Fitzgerald, B. *The Status of the Information Systems Field: Historical Perspective and Practical Orientation*, Information Systems Research, vol. 5, no. 4, 2000.
4. Adler, P. S. and Winograd, T. A. *Usability - Turning Technologies into Tools*. Oxford University Press, New York, USA, 1992.
5. Agarwal, R., Bruno, G. and Torchiano, M. *Instance Modelling - Beyond Object-Oriented Modelling,* 3rd International Conference on Information Technology, Bhubaneswar, India, 2000.
6. Agostini, A., De Michelis, G. and Grasso, M. A. *Rethinking CSCW systems: the architecture of MILANO*, ECSCW Conference, Lancaster, UK, 1997.
7. Agostini, A. and DeMichelis, G. *A Light Workflow Management System Using Simple Process Models*, Computer Supported Cooperative Work, vol. 9, no. 3-4, 2000.
8. Agostini, A., Michelis, G. D. and Loregian, M. *Undo in Workflow Management Systems,* in *Business Process Management 2003, LNCS 2678*, Springer, Berlin, Germany, 2003.
9. Agre, P. *Accountability and Discipline: A Comment on Suchman and Winograd*, Computer Supported Cooperative Work, vol. 3, no. 1, 1995.
10. Alexander, C. *The Origins of Pattern Theory: The Future of the Theory, and the Generation of a Living World*, IEEE Software, vol. 16, no. 5, 1999.
11. Alloui, I., Cimpan, S., Oquendo, F. and Verjus, H. *A Software Framework for Software-Intensive Process Modelling, Enactment and Fuzzy Control*, Transactions of the Society for Design and Process Science, vol. 5, no. 4, 2001.
12. Alonso, G., Fiedler, U., Hagen, C., Lazcano, A., Schuldt, H. and Weiler, N. *WISE: Business to Business E-Commerce,* International Workshop on Research Issues on Data Engineering, Sydney, Australia, 1999.
13. Althoff, K.-D., Nick, M. and Tautz, C. *Improving Organizational Memories Through User Feedback*, Workshop on Learning Software Organizations, Conference on Software Engineering and Knowledge Engineering (SEKE), Kaiserslautern, Germany, 1999.
14. Ambriola, V., Conradi, R. and Fuggetta, A. *Assessing Process-Centered Software Engineering Environments*, ACM Transactions on Software Engineering and Methodology, vol. 6, no. 3, 1997.
15. Andersen, R. *A Configuration Management Approach for Supporting Cooperative Information System Development*, PhD-thesis, The Norwegian Institute of Technology, Trondheim, Norway, 1994.
16. Antunes, P. and Guimaraes, N. *Beyond Formal Processes: Augmenting Workflow with Group Interaction Techniques*, ACM Conference on Organizational Computing Systems (COOCS), Milpitas, California, USA, 1995.
17. Apache Software Foundation *Jakarta Slide homepage, http://jakarta.apache.org/slide,* 2002.
18. Araujo, R. and Borges, M. *Extending the Software Process Culture - An Approach Based on Groupware and Workflow*, in *PROFES Conference, LNCS 2188*. Springer, Berlin, Germany, 2001.
19. Argyris, C. and Schön, D. *Organizational Learning: A Theory of Action Perspective*. Addison Wesley, Reading, MA, USA, 1978.
20. Argyris, C. and Schön, D. *Organizational Learning II: Theory, Method and Practice*. Addison Wesley, Reading, MA, USA, 1996.
21. Arkin, A. *Business Process Modelling Language - BPML 1.0 Working Draft*, BPMI.org, 2002.
22. Armour, P. G. *The Case for a New Business Model*, Communications of the ACM, vol. 43, no. 8, 2000.
23. Atnafu, S., Chbeir, R. and Brunie, L. *Efficient Content-Based and Metadata Retrieval in Image Databases*, IKNOW Conference, Graz, Austria, 2002.

24. Augeraud, M. and Freeman-Benson, B. N. *Dynamic Objects*, ACM Conference on Organizational Computing Systems (COOCS), Milpitas, California, USA, 1993.
25. Avison, D., Lau, F., Myers, M. and Nielsen, P. A. *Action research*, Communications of the ACM, vol. 42, no. 1, 1999.
26. Badouel, E. and Olivier, J. *Reconfigurable Nets, a Class of High Level Petri Nets Supporting Dynamic Changes within Workflow Systems*, Technical Report 3339, INRIA, Rennes, France, 1998.
27. Bancroft, N. H., Seip, H. and Sprengel, A. *Implementing SAP R/3: How to Introduce a Large System into a Large Organisation*. Manning, 1998.
28. Bandinelli, S., Fuggetta, A., Lavazza, L., Loi, M. and Picco, G. P. *Modeling and Improving an Industrial Software Process*, IEEE Transactions on Software Engineering, vol. 21, no. 5, 1995.
29. Bannon, L. *Editorial: Commentaries and a Response in the Suchman-Winograd Debate*, Computer Supported Cooperative Work, vol. 3, no. 1, 1995.
30. Bannon, L. and Bødker, S. *Constructing Common Information Spaces*, ECSCW Conference, Lancaster, England, 1997.
31. Bannon, L. J. *The Politics of Design: Representing Work*, Communications of the ACM, vol. 38, no. 9, 1995.
32. Bannon, L. J. and Schmidt, K. *CSCW: Four characters in search of a context*, ECSCW Conference, Gatwick, U.K., 1989.
33. Bansler, J. P. and Bødker, K. *A Reapprailsal of Structured Analysis: Design in an Organisational Context*, ACM Transactions on Information Systems, vol. 11, no. 2, 1993.
34. Bardram, J. E. *Plans as Situated Action: An Activity Theory Approach to Workflow Systems*, ECSCW Conference, Lancaster, England, 1997.
35. Barlow, C. M. *Exploring a Human Centered Perspective on Collaboration and Knowledge Management Systems*, SSGRR Conference, Rome, Italy, 2001.
36. Barnard, P., May, J., Duke, D. and Duce, D. *Systems, Interaction, and Macrotheory*, ACM Transactions on Computer-Human Interaction, vol. 7, no. 2, 2000.
37. Barth, T. and Eriksen, T. H. *Embodiment and Time: Inventions of Time in Sensory, Instrumental and Semiotic Technologies*, 2nd Nordic Baltic Conference on Activity Theory and Socio-Cultural Research, Rönneby, Sweden, 2002.
38. Baskerville, R. L. *Investigating Information Systems with Action Research*, Communications of the Association for Information Systems, vol. 2, no. 19, 1999.
39. Baumard, P. *Tacit Knowledge in Organizations*. Sage, London, UK, 1999.
40. Benson, I., Everhard, S., McKernan, A., Galewsky, B. and Partridge, C. *Mathematical Structures for Reasoning about Emergent Organization*, ACM CSCW Workshop: Beyond Workflow Management, Philadelphia, USA, 2000.
41. Bentley, R. and Dourish, P. *Medium versus mechanism: Supporting collaboration through customisation*, ECSCW Conference, Stockholm, Sweden, 1995.
42. Bentley, R., Horstmann, T., Sikkel, K. and Trevor, J. *Supporting collaborative information sharing with the World-Wide Web: The BSCW Shared Workspace system*, WWW Conference, Boston, 1995.
43. Berger, P. L. and Luckmann, T. *The Social Construction of Reality. A Treatise in the Sociology of Knowledge*. Penguin Books, USA, 1966.
44. Bergmans, L. and Aksit, M. *Composing Crosscutting Concerns Using Composition Filters*, Communications of the ACM, vol. 44, no. 10, 2001.
45. Bernstein, A. *How Can Cooperative Work Tools Support Dynamic Group Processes? Bridging the Specificity Frontier*, ACM CSCW Conference, Philadelphia, USA, 2000.
46. Bernstein, A. and Jablonski, S. *Workshop: Beyond Workflow Management: Supporting Dynamic Organisational Processes*, ACM CSCW Conference, Philadelphia, USA, 2000.
47. Bernstein, A., Klein, M. and Malone, T. *The Process Recombinator: A Tool for Generating New Business Process Ideas*, ICIS Conference, Charlotte, NC, USA, 1999.
48. Berre, A.-J. *An Object-oriented Framework for Systems Integration and Interoperability*, PhD-thesis, The Norwegian Institute of Technology, Trondheim, Norway, 1993.
49. Bertino, E., Ferrari, E. and Atluri, V. *Specification and Enforcement of Authorization Constraints in Workflow Management Systems*, ACM Transactions on Information and System Security, vol. 2, no. 1, 1999.
50. Bier, E., Stone, M., Pier, K., Buxton, W. and DeRose, T. *Toolglass and Magic Lenses: The See-Through Interface*, SIGGRAPH Conference, Anaheim, California, 1993.

51. Bijker, W. E., Hughes, T. P. and Pinch, T. *The Social Construction of Technological Systems*. MIT Press, Cambridge, Mass., 1987.
52. Birk, A., Dingsøyr, T. and Stålhane, T. *Postmortem: Never Leave a Project without It*, IEEE Software, vol. 19, no. 3, 2002.
53. Blackler, F. *Knowledge, Knowledge Work and Organizations: An Overview and Interpretation*, Organization Studies, 1995.
54. Blauner, R. *Alienation and Freedom*. The University of Chicago Press, Chicago, 1964.
55. Block, N. *Holism, Mental and Semantic*, in *Routledge Encyclopedia of Philosophy*. Routledge, New York, Ny, USA, 1998.
56. Boehm, B. *Get Ready for Agile Methods, with Care*, IEEE Computer, vol. 35, no. 1, 2002.
57. Boehm, B. and Basili, V. R. *Gaining Intellectual Control of Software Development*, IEEE Computer, vol. 33, no. 5, 2000.
58. Bogia, D. P. and Kaplan, S. M. *Flexibility and Control for Dynamic Workflows in the wOrlds Environment*, ACM Conference on Organizational Computing Systems (COOCS), Milpitas, California, 1995.
59. Bogia, D. P., Tolone, W. J., Kaplan, S. M. and Tribouille, E. *Supporting dynamic interdependencies among collaborative activities*, ACM Conference on Organizational Computing Systems (COOCS), 1993.
60. Boland, R. J. J. and Tenkasi, R. V. *Perspective Making and Perspective Taking in Communities of Knowing*, Organization Science, vol. 6, no. 4, 1995.
61. Bolcer, G. and Kaiser, G. *SWAP: Leveraging the Web to Manage Workflow*, IEEE Internet Computing, vol. 3, no. 1, 1999.
62. Bolcer, G. A. and Taylor, R. N. *Endeavors: A Process System Integration Infrastructure*, International Conference on Software Process (ICSP4), Brighton, U.K., 1996.
63. Booch, G., Jacobson, I. and Rumbaugh, J. *The Unified Modeling Language User Guide*. Addison Wesley, Reading, Massachusetts, 1999.
64. Borghoff, U. M. and (Editors), R. P. *Information Technology for Knowledge Management*. Springer Verlag, Berlin, 1998.
65. Borghoff, U. M., Bottoni, P., Mussio, P. and Pareschi, R. *Reflective Agents for Adaptive Workflows*, Practical Aspects of Knowledge Management (PAKM) Conference, Basel, Switzerland, 1996.
66. Borgida, A. and Murata, T. *Workflows as Persistent Objects with Persistent Exceptions - A Framework for Flexibility*, ACM CSCW Workshop: Towards Adaptive Workflow Systems, Seattle, 1998.
67. Bose, P. and Zhou, X. *WWAC: WinWin Abstraction Based Decision Coordination*, ACM Work Activities Coordination and Collaboration Conference (WACC), San Francisco, USA, 1999.
68. Botha, R. A. and Eloff, J. H. P. *A Framework for Access Control in Workflow Systems*, Information Management and Computer Security, vol. 9, no. 3, 2000.
69. Bowers, Button and Sharrock *Workflows from within and without*, ECSCW Conference, Stockholm, Sweden, 1995.
70. Brasethvik, T. and Gulla, J. A. *A Conceptual Modelling Approach to Semantic Document Retrieval*, CAiSE Conference, Springer LNCS 2348, Toronto, Canada, 2002.
71. Brathaug, T. A. and Evjen, T. Å. *Enterprise Modelling*, Technical Report STF 38 A96302, SINTEF, Trondheim, Norway, 1996.
72. Brataas, G. *Integrating Management of Human and Computer Resources in Task Processing Organizations: A Conceptual View*, PhD-thesis, Norwegian Institute of Technology, Trondheim, Norway, 1996.
73. Braverman, M. *Labor and Monopoly Capital*. The University of Chicago Press, Chicago, USA, 1974.
74. Brinkkemper, S. *Method Engineering with Web-Enabled Methods*, in *Information Systems Engineering - State of the Art and Research Themes*, S. Brinkkemper, E. Lindencrona, and A. Sølvberg, Eds. Springer, Berlin, Germany, 2000.
75. Brooks, F. P. jr. *The Mythical Man-Month (20th Anniversary Edition 1995)*. Addison-Wesley, 1975.
76. Brooks, F. P. jr. *No Silver Bullet.: Essence and Accidents of Software Engineering*, Information Processing Conference, 1986.
77. Brown, J. S. and Duguid, P. *Organizational Learning and Communities-of-Practice: Toward a Unified View of Working, Learning and Innovation*, Organization Science, vol. 2, no. 1, 1991.
78. Brown, W. J., McCormick, H. W. Jr. and Thomas, S. W. *Anti-Patterns in Project Management*. John Wiley & Sons, USA, 2000.

79. Bruno, G., Torchiano, M. and Agarwal, R. *Instance modeling - beyond object-oriented modeling*, Conference on Information Technology (CIT), Bhubaneswar, India, 2000.
80. Brunsson, N. *The Organization of Hypocricy, Talk, Decisions and Actions in Organizations*. John Wiely and Sons, New York, 1989.
81. Bunge, M. *Philosophy of Science - Vol.1 From Problem to Theory*, Revised edition, Transaction Publishers, New Brunswick, USA, 1998.
82. Bunge, M. *Philosophy of Science - Vol.2 From Explanation to Justification*, Revised edition, Transaction Publishers, New Brunswick, USA, 1998.
83. Burg, J. F. M. *Linguistic Instruments in Requirements Engineering*, PhD-thesis, Vrije Universiteit, Amsterdam, NL, 1997.
84. Bussler, C. *Process Model Inheritance*, CAiSE Conference, Springer LNCS 2348, Toronto, Canada, 2002.
85. Button, G. *What's Wrong with Speach Act Theory*, Computer Supported Cooperative Work, vol. 3, no. 1, 1995.
86. Cameron, J. *Configurable Development Processes*, Communications of the ACM, vol. 45, no. 3, 2002.
87. Carchiolo, V., D'Ambra, S., Longheu, A. and Malgeri, M. *Object-Oriented Approach in Production Flow Modeling*, Workflow Management Conference, Münster, Germany, 1999.
88. Carlsen, A., Syed, J. and Välikangas, L. *Emerging Characteristics of Knowledge Intensive Work*, Report D99-2184, SRI Consulting, Business Intelligence Program, Menlo Park, CA, USA, 1999.
89. Carlsen, S. *Comprehensible Business Process Models for Process Improvement and Process Support*, CAiSE Doctoral Consortium, Heraklion, Greece, 1996.
90. Carlsen, S. *Conceptual Modeling and Composition of Flexible Workflow Models*, PhD-thesis, Norwegian University of Science and Technology, Trondheim, Norway, 1997.
91. Carlsen, S. *Action Port Model: A Mixed Paradigm Conceptual Workflow Modeling Language*, CoopIS Conference, New York, 1998.
92. Carlsen, S. and Gjersvik, R. *Organizational Metaphors as Lenses for Analyzing Workflow Technology*, ACM GROUP Conference, Phoenix, Arizona USA, 1997.
93. Carlsen, S., Johnsen, S. G., Jørgensen, H. D., Coll, G. J., Mæhle, Å., Carlsen, A. and Hatling, M. *Knowledge re-activation mediated through knowledge carriers*, MICT Conference, Copenhagen, Denmark, 1999.
94. Carlsen, S. and Jørgensen, H. *Emergent Workflow: The AIS Workware Demonstrator*, ACM CSCW Workshop: Towards Adaptive Workflow Systems, Seattle, 1998.
95. Carlsen, S., Jørgensen, H. D., Krogstie, J. and Sølvberg, A. *Process Models as a Knowledge Creation Arena*, EURAM Conference, Stockholm, Sweden, 2002.
96. Carlsen, S., Krogstie, J., Sølvberg, A. and Lindland, O. I. *Evaluating Flexible Workflow Systems*, Hawaii International Conference on System Sciences (HICSS-30), Maui, Hawaii, 1997.
97. Casati, F. *A Discussion on Approaches to Handling Exceptions in Workflows*, ACM CSCW Workshop: Towards Adaptive Workflow Systems, Seattle, 1998.
98. Casati, F., Ceri, S., Pernici, B. and Pozzi, G. *Workflow Evolution*, ER Conference, Cottbus, Germany, 1996.
99. Casati, F. and Shan, M. C. *Dynamic and Adaptive Composition of e-Services*, Information Systems Journal, vol. 26, no. 3, 2001.
100. Castel, F. *Theory, Theory on the Wall ...*, Communications of the ACM, vol. 45, no. 12, 2002.
101. Chen, P. P., Akoka, J., Kangassalo, H. and Thalheim, B. *Conceptual Modeling. Current Issues and Future Directions*. Springer LNCS 1565, Berlin, Germany, 1999.
102. Chen, P. P., Thalheim, B. and Wong, L. Y. *Future Directions of Conceptual Modeling*, in *Conceptual Modelling*, P. P. Chen, J. Akoka, H. Kangassalo, and B. Thalheim, Eds. Springer LNCS 1565, Berlin, Germany, 1999.
103. Chiu, D., Li, Q. and Karlapalem, K. *A Logical Framework for Exception Handling in ADOME Workflow Management System*, in *CAiSE Conference, Springer LNCS 1789*, B. Wangler and L. Bergman, Eds., Stockholm, Sweden, 2000.
104. Chou, S.-C. *ProActNet: Modelling Processes through Activity Networks*, International Journal of Software Engineering and Knowledge Engineering, vol. 12, no. 5, 2002.
105. Chrysostalis, M., Hildrum, J., Krogstie, J., Scagno, G. and Strømseng, K. *Use Case Evaluation Report*, Deliverable 9-93-S-2003-01-2, The EXTERNAL Project, 2003.

106. Clarke, P. and Cooper, M. *Knowledge Management and Collaboration*, Practical Applications of Knowledge Management (PAKM) Conference, Basel, Switzerland, 2000.
107. Clegg, S. *Frameworks of Power*. Sage, London, UK, 1989.
108. Cockburn, A. *Characterizing People as Non-Linear, First-Order Components in Software Development*, Systemics, Cybernetics and Informatics (SCI) Conference, Orlando, Florida, 2000.
109. Cockburn, A. *Selecting a Project's Methodology*, IEEE Software, vol. 17, no. 4, 2000.
110. Cockburn, A. *People and Methodologies in Software Development*, PhD-thesis, University of Oslo, 2003.
111. Colwell, B. *Ground Bounce*, IEEE Computer, vol. 36, no. 3, 2003.
112. Conklin, J. *Wicked Problems and Fragmentation*, in *Dialog Mapping: Defragmenting Projects with Shared Understanding*. cognexus.org, 2003.
113. Conklin, J. and Begeman, L. *gIBIS: A Hypertext Tool for Exploratory Policy Discussion*, ACM Transactions on Office Information Systems, vol. 6, no. 4, 1988.
114. Conradi, R. and Fuggetta, A. *Improving Software Process Improvement*, IEEE Software, vol. 19, no. 4, 2002.
115. Conradi, R. and Jaccheri, M. L. *Process Modelling Languages*, in *Software Process: Principles, Methodology and Technology, Lecture Notes in Computer Science*, Springer LNCS 1500, 1998.
116. Corbett, J. M. *Work at the Interface: Advanced Manufacturing Technology and Job Design*, in *Usability - Turning Technologies into Tools*, P. S. Adler and T. A. Winograd, Eds. Oxford University Press, New York, USA, 1992.
117. Cugola, G. *Tolerating Deviations in Process Support Systems via Flexible Enactment of Process Models*, IEEE Transactions on Software Engineering, vol. 24, no. 11, 1998.
118. Cugola, G., Nitto, E. D., Fuggetta, A. and Ghezzi, C. *A Framework for Formalizing Inconsistencies and Deviations in Human-Centered Systems*, ACM Transactions Software Engineering and Methodology, vol. 5, no. 3, 1996.
119. Curtis, B., Kellner, M. I. and Over, J. *Process Modeling*, Communications of the ACM, vol. 35, no. 9, 1992.
120. Dangelmaier, W., Hamoudia, H. and Klahold, R. F. *Domain Preferences for End-User Tailoring in Shared Workflow Interfaces*, CRIWG Workshop, Darmstadt, Germany, 2002.
121. Davenport, T. H. *Process Innovation*. Harvard Business School Press, Boston, Massachusetts, 1993.
122. Davenport, T. H. and Prusak, L. *Working Knowledge*. Harvard Business School Press, Boston, Massachusetts, 1993.
123. De Michelis, G., Dubois, E., Jarke, M., Matthes, F., Mylopoulos, J., Pohl, K., Schmidt, J., Woo, C. and Yu, E. *Cooperative Informations Systems: A Manifesto*, in *Cooperative Information Systems: Trends and Directions*, M. Papazoglou and G. Schlageter, Eds. Academic Press, 1998.
124. De Michelis, G. and Grasso, M. A. *Situating Conversations within the Language/Action Perspective: The Milan Conversation Model*, ACM CSCW Conference, Chapel Hill, North Carolina, USA, 1994.
125. Dehli, E., Smith-Meyer, H. and Lillehagen, F. *Metis LEARN - Leveraging Enterprise Architecture Repository*, Concurrent Engineering (CE) Conference, Madeira, Portugal, 2003.
126. Delphi Group *BPM 2001 - In Process. The Changing Role of Business Process Management in Today's Economy*, Whitepaper, 2001.
127. DeMarco, T. and Boehm, B. *The Agile Methods Fray*, IEEE Computer, vol. 35, no. 6, 2002.
128. Denning, P. *Computing the Profession*, in *Computer Science and Engineering Education*, T. Greening, Ed. George Mason University, Fairfax, VA, 1998.
129. Denning, P. *Computer Science: The Discipline*, in *Encyclopedia of Computer Science*, A. Ralston and D. Hemmendinger, Eds. George Mason University, Fairfax, VA, 2000.
130. Denning, P. and Dargan, P. A. *Action-Centered Design*, in *Bringing Design to Software*, T. Winograd, Ed. Addison-Wesley, New York, 1996.
131. Denning, P. and Medina-Mora, R. *Completing the Loops*, ACM Interactions, vol. 25, no. 3, 1995.
132. Denning, P. *Flatlined*, Communications of the ACM, vol. 45, no. 6, 2002.
133. Derniame, J. C. *Software Process: Principles, Methodology and Technology*. Springer LNCS 1500, Berlin, Germany, 1998.
134. Dewan, P. *An Integrated Approach to Designing and Evaluating Collaborative Applications and Infrastructures*, Computer Supported Cooperative Work, vol. 10, no. 1, 2001.
135. Divitini, M. and Simone, C. *Supporting Different Dimensions of Adaptability in Workflow Modeling*, Computer Supported Cooperative Work, vol. 9, no. 3-4, 2000.

136. Donath, J. *A Semantic Approach to Visualizing Online Conversations*, Communications of the ACM, vol. 45, no. 4, 2002.
137. Dori, D. *Why Significant UML Change is Unlikely*, Communications of the ACM, vol. 45, no. 11, 2002.
138. Dourish, P. *Developing a Reflective Model of Collaborative Systems*, ACM Transactions on Computer-Human Interaction, vol. 2, no. 1, 1995.
139. Dourish, P. *Utilising Metalevel Techniques in a Flexible Toolkit for CSCW Applications*, ACM Transactions on Computer-Human Interaction, vol. 5, no. 2, 1998.
140. Dourish, P. *Process Descriptions as Organizational Accounting Devices: The Dual Use of Workflow Technologies*, ACM GROUP Conference, Boulder, USA, 2001.
141. Dourish, P. and Bellotti, V. *Awareness and Coordination in Shared Workspaces,* ACM CSCW Conference, Toronto, Canada, 1992.
142. Dourish, P., Holmes, J., MacLean, A., Marqvardsen, P. and Zbyslaw, A. *Freeflow: Mediating Between Representation and Action in Workflow Systems*, ACM CSCW Conference, Boston, USA, 1996.
143. Dowson, M. and Fernstrom, C. *Towards Requirements for Enactment Mechanisms*, 3rd European Workshop on Software Process Technology, 1994.
144. D'Souza, D., Sane, A. and Birchenough, A. *First Class Extensibility for UML - Packaging of Profiles, Stereotypes, Patterns*, UML Conference, Springer LNCS 1723, Fort Collins, USA, 1999.
145. Duddy, K. *UML2 Must Enable a Family of Languages*, Communications of the ACM, vol. 45, no. 11, 2002.
146. Dustdar, S. *Collaborative Knowledge Flow - Improving Process-Awareness and Traceability of Work Activities*, PAKM Conference, Springer LNCS 2569, 2002.
147. Eder, J. and Panagos, E. *Towards Distributed Workflow Process Management*, ACM WACC Workshop: Cross-Organisational Workflow Management and Co-ordination, San Fransisco, USA, 1999.
148. Edmonds, D. and ter Hofstede, A. H. M. *Achieving Workflow Adaptability by Means of Reflection*, ACM CSCW Workshop: Towards Adaptive Workflow Systems, Seattle, 1998.
149. Edwards, W. K. *Policies and Roles in Collaborative Applications*, ACM CSCW Conference, Boston, Mass, 1996.
150. Ehn, P. *Scandinavian Design: On Participation and Skill*, in *Usability - Turning Technologies into Tools*, P. S. Adler and T. A. Winograd, Eds. Oxford University Press, New York, USA, 1992.
151. Eischen, K. *Software Development: an Outsider's View*, IEEE Computer, vol. 35, no. 5, 2002.
152. Ellis, C. and Keddara, K. *ML-DEWS: Modeling Language to Support Dynamic Evolution within Workflow Systems*, Computer Supported Cooperative Work, vol. 9, no. 3-4, 2000.
153. Ellis, C., Keddara, K. and Rozenberg, G. *Dynamic Change Within Workflow Systems*, ACM Conference on Organizational Computing Systems (COOCS), Milpitas, CA, USA, 1995.
154. Ellis, C. and Nutt, G. *Workflow: The Process Spectrum*, NSF Workshop on Workflow and Process Automation in Information Systems, Athens, Georgia, USA, 1996.
155. Ellis, C. A. and Keddara, K. *A Workflow Change is a Workflow*, in *Business Process Management*, W. v. d. Aalst, J. Desel, and A. Oberweis, Eds. Springer LNCS 1806, Berlin, Germany, 2000.
156. Ellis, C. S. and Maltzahn, C. *The Chautauqua Workflow System*, Hawaii International Conference on System Sciences (HICSS-30), Maui, Hawaii, 1997.
157. Elrad, T., Filman, R. E. and Bader, A. *Aspect Oriented Programming - Special Issue*, Communications of the ACM, vol. 44, no. 10, 2001.
158. Emmerich, W., Finkelstein, A., Fugetta, A., Montanegro, C. and Derniame, J.-C. *Software Process - Standards, Assessment and Improvement*, in *Software Process: Principles, Methodology and Technology, Lecture Notes in Computer Science*, Springer LNCS 1500, 1998.
159. Eppler, M. J.-v. , Seifried, P. M. and Röpnack, A. *Improving Knowledge Intensive Processes through an Enterprise Knowledge Medium*, ACM SIGCPR Conference, New Orleans, USA, 1999.
160. Everdingen, Y. v., Hillegersberg, J. v. and Waarts, E. *ERP Adoption by European Midsize Companies*, Communications of the ACM, vol. 43, no. 4, 2000.
161. EXTERNAL *EXTERNAL - Extended Enterprise Resources, Networks And Learning*, EU Project, IST-1999-10091, *New Methods of Work and Electronic Commerce, Dynamic Networked Organisations*. Partners: DNV, GMD-IPSI, Zeus E.E.I.G., METIS, SINTEF Telecom and Informatics, http://www.external-ist.org/, 2000-2002.

162. Falkenberg, E. D., Hesse, W., Lindgreen, P., Nilsson, B. E., Oei, J. L. H., Rolland, C., Stamper, R. K., Assche, F. J. M. V., Verrijn-Stuart, A. A. and Voss, K. *A Framework of Information System Concepts - The FRISCO Report*, IFIP WG 8.1, 1996.
163. Farshchian, B. A. *Integrating Geographically Distributed Development Teams through Increased Product Awareness*, Information Systems Journal, vol. 26, no. 3, 2001.
164. Faustmann, G. *Configuration for Adaptation - A Human-centered Approach to Flexible Workflow Enactment*, Computer Supported Cooperative Work, vol. 9, no. 3-4, 2000.
165. Faustmann, G. and Wikarski, D. *Exception Handling in Petri-Net-Based Workflow Management*, Practical Aspects of Knowledge Management (PAKM) Conference, Basel, Switzerland, 1996.
166. Fekete, A. *Preparation for Research: Instruction in Interpreting and Evaluating Research*, ACM SIGCSE Conference, Philadelphia, USA, 1996.
167. Fischer, L. *Excellence in Practice IV - Innovation and Excellence in Workflow and Knowledge Management*. Workflow Management Coalition, Future Strategies Inc., Florida, USA, 2000.
168. Fox, M. S. and Gruninger, M. *Enterprise Modelling*, AI Magazine, 2000.
169. Franch, X. and Ribo, J. M. *Using UML for Modelling the Static Part of a Software Process*, UML Conference, Springer LNCS 1723, Fort Collins, CO, USA, 1999.
170. Frank, W. and Tyson, K. P. *Be Clear, Clean and Concise*, Communications of the ACM, vol. 45, no. 11, 2002.
171. Freeman, P. A. *Effective Computer Science*, ACM Computing Surveys, vol. 27, no. 1, 1995.
172. Fuggetta, A. and Jaccheri, M. L. *Dynamic Partitioning of Complex Process Models*, Information and Software Technology, vol. 42, 2000.
173. Gamma, E., Helm, R., Johnson, R. and Vlissides, J. *Design Patterns - Elements of Reusable Object-Oriented Software*. Addison-Wesley, Reading, MA, 1994.
174. Gane, C. and Sarson, T. *Structured Systems Analysis: Tools and Techniques*. Prentice Hall, 1979.
175. Gary, K., Lindquist, T. and Koehnemann, H. *Automated Process Support for Organizational and Personal Processes*, ACM GROUP Conference, Phoenix, Arizona USA, 1997.
176. Gasser, L. *The integration of computing and routine work*, ACM Transactions on Office Information Systems, vol. 4, no. 3, 1986.
177. Georgakopoulos, D., Hornick, M. and Sheth, A. *An Overview of Workflow Management: From Process Modeling to Workflow Automation Infrastructure*, Distributed and Parallel Databases, vol. 3, 1995.
178. Gerson, E. M. and Star, S. L. *Analyzing Due Process in the Workplace*, ACM Transactions on Office Information Systems, vol. 4, no. 3, 1986.
179. Ghannouchi, S. A., Jamel, L. and Ghezala, H. H. B. *Contribution of Case-Based Reasoning to Software Processes*, Systemics, Cybernetics and Informatics (SCI) Conference, Orlando, Florida, 2000.
180. Giotopoulos, K., Vassiliadis, B. and Scagno, G. *WP6 Worktop specification*, Deliverable 6-63-W-2001-01-1, The EXTERNAL Project, 2001.
181. Gjersvik, R. *The Modelling Conference: Participation and Visualization for a Process Oriented Intranet*, Technical Report STF38 S00910, SINTEF, Trondheim, Norway, 2000.
182. Gjersvik, R. and Hepsø, V. *Using Models of Work Practice as Reflective and Communicative Devices: Two Cases from the Norwegian Offshore Industry*, Participatory Design Conference, 1998.
183. Glance, N. S., Pagani, D. S. and Pareschi, R. *Generalized Process Structure Grammars (GPSG) for Flexible Representation of Work*, ACM CSCW Conference, Boston, USA, 1996.
184. Glass, R. L. *A Story about the Crativity Involved in Software Work*, IEEE Software, vol. 18, no. 5, 2001.
185. Glass, R. L. *Searching for the Holy Grail of Software Engineering*, Communications of the ACM, vol. 45, no. 6, 2002.
186. Gnatz, M., Marschall, F., Popp, G., Rausch, A. and Schwerin, W. *Towards a Living Software Development Process Based on Process Patterns*, EWSPT Conference, Springer LNCS 2077, 2001.
187. Goebl, W., Gruber, R., Messner, K., Schwarzer, B., List, B. and Quirchmayr, G. *Introducing Workflow Management Systems in Insurance Companies*, Systemics, Cybernetics and Informatics (SCI) Conference, Orlando, Florida, 2000.
188. Goguen, J. A. *On Notation*, University of California, San Diego, Computer Science and Engineering Department, http://www-cse.ucsd.edu/users/goguen/pubs/, USA, 1993.
189. Goguen, J. A. *Formality and Informality in Requirements Engineering*, Fourth International Conference on Requirements Engineering, 1996.

190. Goguen, J. A. *Towards a Social, Ethical Theory of Information*, in *Social Science Research, Technical Systems and Cooperative Work: Beyond the Great Divide*, G. Bowker, L. Gasser, L. Star, and W. Turner, Eds. Erlbaum, 1997.
191. Gosling, J., Joy, B. and Steele, G. *The Java Language Specification*. Addison-Wesley, 1996.
192. Grasso, A., Meunier, J.-L., Pagani, D. and Pareschi, R. *Distributed Coordination and Workflow on the World Wide Web*, Computer Supported Cooperative Work, vol. 6, 1997.
193. Gray, J., Bapty, T., Neema, S. and Tuck, J. *Handling Crosscutting Constraints in Domain-Specific Modeling*, Communications of the ACM, vol. 44, no. 10, 2001.
194. Green, D. A. and Newth, D. *Towards a Theory of Everything? - Grand Challenges in Complexity and Informatics*, Complexity International, vol. 8, 2001.
195. Green, P. and Rosemann, M. *Integrated Process Modeling: An Ontolocial Evaluation*, Information Systems, vol. 25, no. 3, 2000.
196. Greenberg, S. and Johnson, B. *Studying Awareness in Contact Facilitation*, ACM CHI Workshop: Awareness in Collaborative Systems, Atlanta, Georgia, 1997.
197. Greenwood, R. M., Balasubrmaniam, D., Kirvy, G., Mayes, K., Morrison, R., Seet, W., Warboys, B. and Zirintsis, E. *Reflection and Reification in Process Systems Evolution: Experience and Opportunity*, EWSPT Conference, Springer LNCS 2077, 2001.
198. Greenwood, R. M., Robertson, I., Snowdon, R. A. and Warboys, B. C. *Active Models in Business*, Conference on Business Information Technology (CBIT), 1995.
199. Grinter, R. E. *Doing Software Development: Occasions for Automation and Formalisation*, ECSCW Conference, Lancaster, UK, 1997.
200. Grinter, R. E. *Workflow Systems: Occasions for Success and Failure*, Computer Supported Cooperative Work, vol. 9, 2000.
201. Grosz, G., Rolland, C., Schwer, S., Souveyet, C., Plihon, V., Si-Said, S., Achour, C. B. and Gnaho, C. *Modelling and Engineering the Requirements Engineering Process : An Overview of the NATURE Approach*, Requirements Engineering, vol. 3, no. 2, 1997.
202. Grudin, J. *Groupware and Social Dynamics - Eight Challenges for Developers*, Communications of the ACM, vol. 37, no. 1, 1994.
203. Grudin, J. and Poltrock, S. E. *Computer-Supported Cooperative Work and Groupware*, in *Advances in Computers*, M. Zelkowitz, Ed. Academic Press, Orlando, 1997.
204. Grundy, J. C., Hosking, J. G. and Mugridge, W. B. *Inconsistency Management for Multiple-View Software Development Environments*, IEEE Transactions on Software Engineering, vol. 24, no. 11, 1998.
205. Gulla, J. A. *A General Explanation Component for Conceptual Modeling in CASE Environments*, ACM Transactions on Information Systems, vol. 14, no. 3, 1996.
206. Gulla, J. A., Lindland, O. I. and Willumsen, G. *PPP - An Integrated CASE Environment*, CAiSE Conference, Trondheim, Norway, 1991.
207. Gutwin, C., Greenberg, S. and Roseman, M. *Workspace Awareness in Real-Time Distributed Groupware: Framework, Widgets, and Evaluation*, HCI Conference, 1996.
208. Hamilton, A. G. *Logic for Mathematicians*. Cambridge University Press, New York, NY, USA, 1978 (revised 1988).
209. Harel, D. *Statecharts: A Visual Formalism for Complex Systems*, Science of Computer Programming, vol. 8, 1987.
210. Hartmanis, J. *On Computational Complexity and the Nature of Computer Science*, Communications of the ACM, vol. 37, no. 10, 1994.
211. Hartmanis, J. *Responses to the Essays "On Computational Complexity and the Nature of Computer Science"*, ACM Computing Surveys, vol. 27, no. 1, 1995.
212. Hayes, N. *Work-arounds and Boundary Crossing in a High Tech Optonics Company: The Role of Co-operative Workflow Technologies*, Computer Supported Cooperative Work, vol. 9, no. 3-4, 2000.
213. Heinl, P., Horn, S., Jablonski, S., Neeb, J., Stein, K. and Teschke, M. *A Comprehensive Approach to Flexibility in Workflow Management Systems*, ACM Work Activities Coordination and Collaboration Conference (WACC), San Francisco, USA, 1999.
214. Herrmann, T. *Evolving Workflows by User-driven Coordination*, DCSCW Conference, München, Germany, 2000.
215. Herrmann, T., Hoffmann, M., Loser, K.-U. and Moysich, K. *Semistructured models are surprisingly useful for user-centered design*, COOP Conference, Sophia Antipolis, France, 2000.

216. Herrmann, T. and Kunau, G. *Costs and Benefits of Structuration*, COOP Conference, München, Germany, 2002.
217. Herrmann, T. and Loser, K.-U. *Vagueness in models of socio-technical systems*, Behaviour & Information Technology, vol. 18, no. 5, 1999.
218. Hewitt, C. *Offices are Open Systems*, ACM Transactions on Office Information Systems, vol. 4, no. 3, 1986.
219. Highsmith, J. and Cockburn, A. *Agile Software Development: The Business of Innovation*, IEEE Computer, vol. 34, no. 9, 2001.
220. Hirschheim, R. A. *Understanding the Office: A Social-Analythic Perspective*, ACM Transactions on Office Information Systems, vol. 4, no. 4, 1986.
221. Hirschheim, R. A. and Klein, H. K. *Four Paradigms of Information Systems Development*, Communications of the ACM, vol. 32, no. 10, 1989.
222. Hitomi, A. S. and Lee, D. *Endeavors and Component Reuse in Web-Driven Process Workflow*, California Software Symposium, 1998.
223. Hofmann, A. *Development of a library of reusable workflow objects*, SIGGROUP Bulletin, vol. 18, no. 1, 1997.
224. Holm, P. and Karlgren, K. *Theories of Meaning and Different Perspectives on Information Systems*, Conference on Information System Concepts (ISCO), Marburg, Germany, 1995.
225. Holt, A. W., Ramsey, H. R. and Grimes, J. D. *Coordination System Technology as the Basis for a Programming Environment*, ITT Technical Journal, vol. 57, no. 4, 1983.
226. Hommes, B.-J. and Reijswoud, V. v. *The Quality of Business Process Modelling Techniques*, Conference on Information Systems Concepts (ISCO), Leiden, Nertherlands, 1999.
227. Hong, W. K. and Madhavji, N. H. *The Role of a Software Process Generaliser in Managing a Line of Products*, 10th International Software Process Workshop, Dijon, France, 1996.
228. Horn, S. and Jablonski, S. *An Approach to Dynamic Instance Adaptation in Workflow Management Applications*, ACM CSCW Workshop: Towards Adaptive Workflow Systems, Seattle, 1998.
229. Hruby, P. *Structuring Specification of Business Systems with UML (with an Emphasis on Workflow Systems)*, OOPSLA Business Object Workshop, 1998.
230. Huhns, M. N. and Singh, M. P. *Workflow Agents*, IEEE Internet Computing, vol. 2, no. 4, 1998.
231. Hull, R., Llirbat, F., Simon, E., Su, J., Dong, G., Kumar, B. and Zhou, G. *Declarative Workflows that Support Easy Modification and Dynamic Browsing*, ACM Work Activities Coordination and Collaboration (WACC) Conference, San Francisco, USA, 1999.
232. Haake, J., Ohren, O. and Krogstie, J. *EXTERNAL WP4 - D13. Prototype: Use case - External Project*, Deliverable 4-00-D-2002-01-0, The EXTERNAL Project, 2002.
233. Haake, J. M. and Wang, W. *Flexible Support for Business Processes: Extending Cooperative Hypermedia with Process Support*, ACM GROUP Conference, Phoenix, Arizona USA, 1997.
234. IDEF-3x *Process Modeling Language Specification*, Standard NA-94-1422B, Rockwell International, 1993.
235. IETF *WebDAV standard, http://www.webdav.org/*
236. Jablonski, S. *MOBILE: A Modular Workflow Model and Architecture*, Conference on Dynamic Modelling and Information Systems, Noordijkerhout, Netherlands, 1994.
237. Jablonski, S. *Workflow Management between Formal Theory and Pragmatic Approaches*, in *Business Process Management*, W. v. d. Aalst, J. Desel, and A. Oberweis, Eds. Springer LNCS 1806, Berlin, Germany, 2000.
238. Jaccheri, M. L. *Reusing Software Process Models in E3*, University of Torino, Italy, 1999.
239. Jaccheri, M. L. and Conradi, R. *Techniques for Process Model Evolution in EPOS*, IEEE Transactions on Software Engineering, vol. 19, no. 12, 1993.
240. Jaccheri, M. L., Conradi, R. and Dyrnes, B. H. *Software Process Technology and Software Organisations*, EWSPT Conference, Springer LNCS 1780, Kaprun, Austria, 2000.
241. Jaccheri, M. L., Picco, G. P. and Lago, P. *Eliciting Software Process Models with the E3 Language*, ACM Transactions on Software Engineering and Methodology, vol. 7, no. 4, 1998.
242. Jackman, H. *Holism, Relevance and Thought Content*, Proceedings of the Ohio Philosophial Association, 1999.
243. Jackman, H. *Moderate Holism and the Instability Thesis*, American Philosophical Quarterly, vol. 36, no. 4, 1999.
244. Jacobson, I. *Object-Oriented Software Engineering: A Use Case Driven Approach*. Addison-Wesley, Wokingham, 1993.

245. Jacobson, I., Ericsson, M. and Jacobson, A. *The Object Advantage: Business Process Reengineering with Object Technology*. Addison-Wesley, Wokingham, 1994.
246. Janssens, G. K., Verelst, J. and Weyn, B. *Techniques for Modelling Workflows and Their Support of Reuse*, in *Business Process Management*, W. v. d. Aalst, J. Desel, and A. Oberweis, Eds. Springer LNCS 1806, Berlin, Germany, 2000.
247. Jarke, M., Pohl, K., Rolland, C. and Schmitt, J.-R. *Experience-Based Method Evaluation and Improvement: A Process Modeling Approach*, IFIP WG 8.1 Conference, Maastricht, Netherlands, 1994.
248. Julsrud, T. E. and Akselsen, S. *Projects as Business-to-Business Collaboration*, Telenor Research and Development, Oslo, Norway, 2001.
249. Jørgensen, H. D. *Software Process Model Reuse and Learning*, Process Support for Distributed Team-based Software Development (PDTSD), Systemics, Cybernetics and Informatics (SCI) Conference, Orlando, Florida, 2000.
250. Jørgensen, H. D. *Supporting Knowledge Work with Emergent Process Models*, ACM CSCW Workshop: Beyond Workflow Management, Philadelphia, USA, 2000.
251. Jørgensen, H. D. *Interaction as a Framework for Flexible Workflow Modelling*, ACM GROUP Conference, Boulder, USA, 2001.
252. Jørgensen, H. D. *Interactive Process Models for Knowledge Intensive Project Work*, CAiSE Doctoral Consortium, Toronto, Canada, 2002.
253. Jørgensen, H. D. *Model-driven work management services*, Concurrent Engineering (CE) Conference, Madeira, Portugal, 2003.
254. Jørgensen, H. D. and Carlsen, S. *Emergent Workflow: Integrated Planning and Performance of Process Instances*, Workflow Management Conference, Münster, Germany, 1999.
255. Jørgensen, H. D. and Carlsen, S. *Writings in Process Knowledge Management*, Technical Report STF40 A00011, SINTEF, Oslo, Norway, 2000.
256. Jørgensen, H. D. and Krogstie, J. *Active Models for Cooperative Information Systems*, Norwegian Computer Science Conference (NIK), Tromsø, Norway, 2001.
257. Jørgensen, H. D., Krogstie, J., Ohren, O. P. and Johnsen, S. G. *Interactive Models for Tailorable and Evolving Information Systems*, accepted by guest editors, Journal of Applied Systems Studies, 2003.
258. Jørgensen, H. D., Paulsen, T., Storvik, J. and Carlsen, S. *Living Knowledge and Knowledge Tools*, Technical Report, SINTEF, Oslo, Norway, 1999.
259. Kahler, H., Mørch, A., Stiemerling, O. and Wulf, V. *Special Issue on Tailorable Systems and Cooperative Work*, Computer Supported Cooperative Work, vol. 9, no. 1, 2000.
260. Kahng, J. and McLeod, D. *Dynamic Classificational Ontologies: Mediation of Information Sharing in Cooperative Federated Database Systems*, in *Cooperative Information Systems: Trends and Directions*, M. Papazoglou and G. Schlageter, Eds. Academic Press, 1998.
261. Kammer, P. J., Bolcer, G. A., Naylor, R. N. and Bergman, M. *Techniques for Supporting Dynamic and Adaptive Workflow*, Computer Supported Cooperative Work, vol. 9, no. 3-4, 2000.
262. Kangassalo, H. *Conceptual Description for Information Modelling Based on Intensional Containment Relation*, Knowledge Representation meet Databases, KRDB, Budapest, Hungary, 1996.
263. Kangassalo, H. *Are Global Understanding, Communication and Information Management in Information Systems Possible?*, in *Conceptual Modeling. Current Issues and Future Directions*, P. P. Chen, J. Akoka, H. Kangassalo, and B. Thalheim, Eds. Springer LNCS 1565, Berlin, Germany, 1999.
264. Karlsen, T. K. *Innføring av samarbeidsteknologi. Prosjektnotat fra AIS - Avansert Intranett Samarbeid (In Norwegian)*, FAFO Institue of Applied Social Science, 1998.
265. Katzenstein, G. and Lerch, F. J. *Beneath the Surface of Organizational Process: A Social Representation Framework for Business Process Redesign*, ACM Transactions on Information Systems, vol. 18, no. 4, 2000.
266. Kelly, K. *New Rules for the New Economy*, Wired, September, 1997.
267. Kensing, F. and Blomberg, J. *Participatory Design: Issues and Concerns*, Computer Supported Cooperative Work, vol. 7, 1998.
268. Kiczales, G. *Beyond the Black Box: Open Implementation*, IEEE Software, vol. 13, no. 1, 1996.
269. Kiczales, G., Lamping, J., Lopes, C. V., Maeda, C., Mendhekar, A. and Murphy, G. *Open Implementation Design Guidelines*, International Conference on Software Engineering (ICSE), Boston, 1997.
270. Kimbrough, S. O. and Moore, S. A. *On Automated Message Processing in Electronic Commerce and Work Support Systems: Speech Act Theory and Expressive Felicity*, ACM Transactions on Information Systems, vol. 15, no. 4, 1997.

271. King, J. L. *SimLanguage*, Computer Supported Cooperative Work, vol. 3, no. 1, 1995.
272. Kjær, A. and Madsen, K. H. *Participatory Analysis of Flexibility*, Communications of the ACM, vol. 38, no. 5, 1995.
273. Klein, H. K. and Myers, M. D. *A Set of Principles for Conducting and Evaluating Interpretive Field Studies in Information Systems*, MIS Quartely, vol. 23, no. 1, 1999.
274. Klein, M. *Workshop: Towards Adaptive Workflow Systems*, ACM CSCW Conference, Seattle, 1998.
275. Klein, M. and Dellarocas, C. *A Knowledge-based Approach to Handling Exceptions in Workflow Systems*, Computer Supported Cooperative Work, vol. 9, no. 3-4, 2000.
276. Klein, M., Dellarocas, C. and Bernstein, A. *Special Issue on Adaptive Workflow Systems*, Computer Supported Cooperative Work, vol. 9, no. 3-4, 2000.
277. Knutilla, A. J., Stevens, M. P. and Allen, R. H. *Workshop on Evaluating Collaborative Enterprises*, WET-ICE, Gaithersburg, Maryland, USA, 2000.
278. Kolawa, A. *Certification Will Do More Harm Than Good*, IEEE Computer, vol. 35, no. 6, 2002.
279. Koma-Sirviö, S., Mäntyniemi, A. and Seppänen, V. *Towards a Practical Solution for Capturing Knowledge for Software Projects*, IEEE Software, vol. 19, no. 3, 2002.
280. Konstantas, D., Morin, J.-H., Barziv, O., Koumpis, A. and Kobel, C. *Active Business Objects (ABOs): A Novel Paradigm for Building (and Using!) Business Information Systems*, in *Internet Objects*, D. Tsichritzis, Ed. Centre Universitaire d'Informatique, Genéve, Switzerland, 2000.
281. Korbyn, C. *UML 2001: A Standardization Odyssey*, Communications of the ACM, vol. 42, no. 10, 1999.
282. Kosoresow, A. P. and Kaiser, G. E. *Using Agents to Enable Collaborative Work*, IEEE Internet Computing, vol. 2, no. 4, 1998.
283. Kreifelts, T., Hinrichs, E. and Woetzel, G. *Sharing To-Do Lists with a Distributed Task Manager*, ECSCW Conference, 1993.
284. Krogstie, J. *Conceptual Modeling for Computerized Information Systems Support in Organizations*, PhD-thesis, The Norwegian Institute of Technology, Trondheim, Norway, 1995.
285. Krogstie, J. et al. *EE Methodology*, Deliverable D2, The EXTERNAL Project, Oslo, Norway, 2001.
286. Krogstie, J. et al. *EE Methodology version 2*, Deliverable D6, The EXTERNAL Project, Oslo, Norway, 2002.
287. Krogstie, J., Hildrum, J., Chrysostalis, M. and Hestvik, R. *Enterprise Methodology Evaluation Report*, Deliverable D19, The EXTERNAL Project, 2002.
288. Krogstie, J. and Jørgensen, H. D. *Flexible Support of Work Processes - Balancing the Support of Organisations and Workers*, CAiSE Conference Panel, Springer LNCS 2068, Interlaken, Switzerland, 2001.
289. Krogstie, J. and Jørgensen, H. D. *Quality of Interactive Models*, ER Workshop on Conceptual Modeling Quality (IWCMQ), Tampere, Finland, 2002.
290. Krogstie, J., Jørgensen, H. D. and Lillehagen, F. *Active Models for Digitally Enabled Creative Networks*, IFIP World Computing Conference, Montreal, Canada, 2002.
291. Krogstie, J., Lindland, O. I. and Sindre, G. *Defining Quality Aspects for Conceptual Models*, IFIP8.1 Conference on Information Systems Concepts (ISCO), Marburg, Germany, 1995.
292. Kruschwitz, N. and Roth, G. *Inventing Organizations of the 21st Century: Producing Learning through Collaboration*, Report CCS WP 207, Massachusetts Institute of Technology, Cambridge, USA, 1999.
293. Kuhn, T. S. *The Structure of Scientific Revolutions*, 2nd ed. University of Chicago Press, USA, 1962.
294. Kumar, K. *ERP Experiences and Evolution: Introduction to Special Issue*, Communications of the ACM, vol. 43, no. 4, 2000.
295. Kuntz, J. C., Christiansen, T. R., Cohen, G. P., Jin, Y. and Levitt, R. E. *The Virtual Design Team: A Computational Simulation Model of Project Organizations*, Communications of the ACM, vol. 41, no. 11, 1998.
296. Kyng, M. *Making Representations Work*, Communications of the ACM, vol. 38, no. 9, 1995.
297. Lakos, C. *From Coloured Petri Nets to Object Petri Nets*, Conference on the Application and Theory of Petri Nets, Torino, Italy, 1995.
298. Lakos, C. *Pragmatic Inheritance Issues for Object Petri Nets*, TOOLS Pacific Conference, Melbourne, Australia, 1995.
299. Larman, C. *Protected Variation: The Importance of Being Closed*, IEEE Software, vol. 18, no. 3, 2001.
300. Latour, B. *Science in Action*. Open University Press, Milton Keynes, 1987.

301. Lawton, G. *Knowledge Management: Ready for Prime Time?*, IEEE Computer, vol. 34, no. 2, 2001.
302. Ledgard, H. F. *The Emperor with no Clothes*, Communications of the ACM, vol. 44, no. 10, 2001.
303. Lee, J. *Goal-Based Process Analysis: A Method for Systematic Process Redesign*, ACM Conference on Organizational Computing Systems (COOCS), Milpitas, CA, USA, 1993.
304. Lee, J., Gruninger, M., Jin, Y., Malone, T., Tate, A. and Yost, G. *The PIF Process Interchange Format and Framework, version 1.1.*, Standard, MIT Center for Coordination Science, 1996.
305. Lei, Y. and Singh, M. P. *A Comparison of Workflow Metamodels*, ER Workshop on Behavioral Modeling, Springer LNCS 1565, 1997.
306. Lethbridge, T. L. *What Knowledge is Important to a Software Professional?*, IEEE Computer, vol. 33, no. 5, 2000.
307. Levin, M. *Technology Transfer is Organizational Development*, International Journal of Technology Management, vol. 14, no. 2/3/4, 1995.
308. Li, D. and Patrao, J. *Demonstrational Customisation of Shared Whiteboard to Support User-Defined Semantic Relationships among Objects*, ACM GROUP Conference, Boulder, USA, 2001.
309. Lillehagen, F. *Visual Extended Enterprise Engineering Embedding Knowledge Management, Systems Engineering and Work Execution*, IFIP International Enterprise Modelling Conference, Verdal, Norway, 1999.
310. Lillehagen, F. *The Foundations of AKM Technology*, Concurrent Engineering (CE) Conference, Madeira, Portugal, 2003.
311. Lillehagen, F., Dehli, E., Fjeld, L., Krogstie, J. and Jørgensen, H. D. *Active Knowledge Models as a Basis for an Infrastructure for Virtual Enterprises*, IFIP Conference on Infrastructures for Virtual Enterprises (PRO-VE), Sesimbra, Portugal, 2002.
312. Lillehagen, F., Krogstie, J., Jørgensen, H. D. and Hildrum, J. *Active Knowledge Models for Supporting eWork and eBusiness*, International Conference on Concurrent Enterprising (ICE), Rome, Italy, 2002.
313. Lindert, F. and Deiters, W. *Modelling Inter-Organizational Processes with Process Model Fragments*, Enterprise-Wide and Cross-Enterprise Workflow Management Workshop, Paderborn, Germany, 1999.
314. Lindvall, M. and Rus, I. *Process Diversity in Software Development*, IEEE Software, vol. 17, no. 4, 2000.
315. Loos, P. and Allweyer, T. *Process Orientation and Object-Orientation - An Approach for Integrating UML with Event-Driven Process Chains (EPC)*, University of Saarland, Germany, 1998.
316. Loui, M. C. *Computer Science Is a New Engineering Discipline*, ACM Computing Surveys, vol. 27, no. 1, 1995.
317. Louridas, P. and Loucopoilos, P. *A Generic Model for Reflective Design*, ACM Transactions on Software Engineering and Methodology, vol. 9, no. 2, 2000.
318. Ludwig, H., Shan, M.-C., Bussler, C. and Grefen, P. *Cross-Organisational Workflow Management and Co-ordination - WACC'99 Workshop Report*, SIGGROUP Bulletin, 1999.
319. Luo, Z. and Sheth, A. *Defeasible Workflow, its Computation and Exception Handling*, ACM CSCW Workshop: Towards Adaptive Workflow Systems, Seattle, 1998.
320. Luo, Z., Sheth, A., Kochut, K. and Miller, J. *Exception Handling in Workflow Systems*, Applied Intelligence, vol. 13, no. 2, 2000.
321. Lysgaard, S. *Arbeiderkollektivet (In Norwegian)*. Universitetsforlaget, Oslo, 1960.
322. Lyytinen, K. and Zhang, Z. *A Framework for Component Reuse in MetaCASE Based Software Development*, in *Information Systems Engineering - State of the Art and Research Themes*, S. Brinkkemper, E. Lindencrona, and A. Sølvberg, Eds. Springer, Berlin, Germany, 2000.
323. Maginnis, T. *Engineer's Don't Build*, IEEE Software, vol. 17, no. 1, 2000.
324. Majchrzak, A., Rice, R. E., Malhotra, A., King, N. and Ba, S. *Technology Adaptation: The Case of a Computer-Supported Inter-Organizational Virtual Team*, MIS Quarterly, vol. 24, no. 4, 2000.
325. Malhotra, Y. *Role of Information Technology in Managing Organizational Change and Organizational Interdependencies*, http://www.brint.com/, 1996.
326. Malone, T. W. *Commentary on Suchman Article and Winograd Response*, Computer Supported Cooperative Work, vol. 3, no. 1, 1995.
327. Malone, T. W. and Crowston, K. *What is Coordination Theory and How Can It Help Design Cooperative Work Systems*, ACM CSCW Conference, 1990.
328. Malone, T. W. and Crowston, K. *The Interdisciplinary Study of Coordination*, ACM Computing Surveys, vol. 26, no. 1, 1994.

329. Malone, T. W., Crowston, K., Lee, J. and Pentland, B. *Tools for inventing organizations: Toward a handbook of organizational processes*, IEEE Workshop on Enabling Technologies Infrastructure for Collaborative Enterprises, Morgantown, USA, 1993.
330. Malone, T. W., Lai, K.-Y. and Fry, C. *Experiments with Oval: A Radically Tailorable Tool for Cooperative Work*, ACM Transactions on Information Systems, vol. 13, no. 2, 1995.
331. Man, J. D. *A Lightweight Process-Centered Project Support Environment: Motivation, Implementation and Experience*, Process Support for Distributed Team-based Software Development (PDTSD'00), SCI Conference, Orlando, Florida, 2000.
332. Manola, F., Georgakopoulos, D., Heiler, S., Hurwitz, B., Mitchell, G. and Nayeri, F. *Supporting Cooperation in Enterprise-Scale Distributed Object Systems*, in *Cooperative Information Systems: Trends and Directions*, M. Papazoglou and G. Schlageter, Eds. Academic Press, 1998.
333. Manolescu, D. A. and Johnson, R. E. *Dynamic Object Model and Adaptive Workflow*, OOPSLA Workshop on Metadata and Active Object Models, 1999.
334. March, J. and Simon, H. *Organizations*. John Wiley, New York, 1958.
335. Marshall, C. *Enterprise Modeling with UML*. Addison-Wesley, 1999.
336. Massey, A. P., Huang, Y.-T. C., Montoya-Weiss, M. and Ramesh, V. *When Culture and Style aren't About Clothes: Perceptions of Task-Technology "Fit" in Global Virtual Teams*, ACM GROUP Conference, Boulder, USA, 2001.
337. Mathiassen, L. *Reflective Systems Development*, Dr. Techn. Thesis, Aalborg University, Denmark, 1998.
338. Mathiassen, L. and Munk-Madsen, A. *Formalization in Systems Development*, Behaviour and Information Technology, vol. 5, no. 2, 1986.
339. Medina-Mora, R., Winograd, T., Flores, R. and Flores, F. *The Action Workflow Approach to Workflow Management Technology*, ACM CSCW Conference, 1992.
340. Mehandjiev, N., Bottaci, L. and Phillips, R. *User Enhancability for Fast Response to Changing Office Needs*, 27th Annual Hawaii International Conference on System Sciences (HICSS-27), 1994.
341. Meijler, T. D., Kessels, H., Vuijst, C. and le Comte, R. *Realising Run-time Adaptable Workflow by means of Reflection in the Baan Workflow Engine*, ACM CSCW Workshop: Towards Adaptive Workflow Systems, Seattle, 1998.
342. Mellor, S. J. *Make Models be Assets*, Communications of the ACM, vol. 45, no. 11, 2002.
343. Mentzas, G. and Apostolou, D. *Managing Corporate Knowledge: A Comparative Analysis of Experiences in Cosulting Firms*, Practical Aspects of Knowledge Management (PAKM) Conference, Basel, Switzerland, 1998.
344. Merlyn, P. R. and Välikangas, L. *From IT to KT - but Don't Forget the User*, Report D98-2180, SRI Consulting, Business Intelligence Program, Menlo Park, USA, 1998.
345. MetaCase *MetaEdit+ Revolutionized the Way NOKIA Develops Mobile Phone Software*, MetaCase Consulting, Jyväskyla, Finland, 1999.
346. MetaCase *ABC to MetaCase Technology*, MetaCase Consulting, Jyväskyla, Finland, 2000.
347. *METIS GEM Template Reference Guide - Version 1.9.1*, J. Høyte, Ed. NCR METIS, Horten, Norway, 1997.
348. Miers, D. *Vectus Version 3*, Evaluation Report 1-Oct-1998 1:1, Enix, UK, 1998.
349. Miers, D. and Hunt, R. *Process Product Watch - Work Management Technologies Report - Evaluation Framework Process Support Systems*, Enix, UK, 1995.
350. Miers, D. and Hutton, G. *The Business Case for Case Handling*, Enix, UK, 1996.
351. Miller, S. K. *Aspect-Oriented Programming Takes Aim at Software Complexity*, IEEE Computer, vol. 34, no. 4, 2001.
352. Mintzberg, H. *A Typology of Organizational Structure*, in *Readings in Groupware and Computer-Supported Cooperative Work*. Morgan Kaufmann, San Mateo, USA, 1993.
353. Mitcham, C. *Thinking through Technology*. The University of Chicago Press, Chicago, 1994.
354. Moldt, D. and Valk, R. *Object Oriented Petri Nets in Business Process Modeling*, in *Business Process Management*, W. v. d. Aalst, J. Desel, and A. Oberweis, Eds. Springer LNCS 1806, Berlin, Germany, 2000.
355. Montanegro, C., Derniame, J.-C., Kaba, A. B. and Warboys, B. *The Software Process: Modelling and Technology*, in *Software Process: Principles, Methodology and Technology*, J.-C. Derniame, Ed. Springer LNCS 1500, Berlin, Germany, 1998.
356. Moody, D. L., Sindre, G., Brasethvik, T. and Sølvberg, A. *Evaluating the Quality of Process Models: Empirical Testing of a Quality Framework*, ER Conference, Springer LNCS 2503, 2002.

357. Morgan, G. *Images of Organization*. Sage, Beverly Hills, 1986.
358. Myers, B. A., McDaniel, R., Miller, R., Ferrency, A. S., Faulring, A., Kyle, B. D., Mickish, A., Klimovitski, A. and Doane, P. *The Amulet Environment: New Models for Effective User Interface Software Development*, IEEE Transactions on Software Engineering, vol. 23, no. 6, 1997.
359. Myers, M. D. *Qualitative Research in Information Systems*in MISQ Discovery, http://www.isworld.org/, 2001.
360. Mühlen, M. z. *Resource Modeling in Workflow Applications*, Workflow Management Conference, Münster, Germany, 1999.
361. Mühlen, M. z. and Becker, J. *Workflow Management and Object-Orientation - A Matter of Perspectives or Why Perspectives Matter*, OOPSLA Workshop on Object-Oriented Workflow Management, Denver, USA, 1999.
362. Mørch, A. and Mehandjiev, N. *Tailoring as Collaboration: The Mediating Role of Multiple Representations and Application Units*, Computer Supported Cooperative Work, vol. 9, no. 1, 2000.
363. Mørch, A., Stiemerling, O. and Wulf, V. *Tailorable Groupware: Issues, Methods and Architectures*, Workshop Report in SIGGROUP Bulletin, vol. 19, no. 1, 1998.
364. Nardi, B. A., Whittaker, S., Isaacs, E., Creech, M., Johnson, J. and Hainsworth, J. *Integrating Communication and Information through ContactMap*, Communications of the ACM, vol. 45, no. 4, 2002.
365. Natvig, M. K. and Ohren, O. *Modelling shared information spaces (SIS)*, ACM GROUP Conference, Phoenix, Arizona USA, 1999.
366. Neeb, J. *Supporting Dynamic Organisational Processes by Administration Workflows*, ACM CSCW Workshop: Beyond Workflow Management, Philadelphia, USA, 2000.
367. Nelson, R. R. and Winter, S. G. *An Evolutionary Theory of Economic Change*. The Belknap Press of Harvard University Press, Cambridge, Massachusetts and London, England, 1982.
368. Nguyen, M. N., Wang, A. I. and Conradi, R. *Total Software Process Model Evolution in EPOS*, ACM ICSE Conference, Boston, USA, 1997.
369. Niinimäki, M. *Intensional and extensional languages in conceptual modelling*, 10th European-Japanese Conference on Information Modelling and Knowledge Bases, Saariselkä, Finland, 2000.
370. Nonaka, I. and Takeuchi, H. *The Knowledge Creating Company: How Japanese Companies Create the Dynamics of Innovation*. Oxford University Press, New York, 1995.
371. Nuseibeh, B., Finkelstein, A. and Kramer, J. *Method Engineering for Multi-Perspective Software Development*, Information and Software Technology, 1996.
372. Nwana, H. S. *Software Agents: An Overview*, Knowledge Engineering Review, vol. 11, no. 3, 1996.
373. Ohren, O. P., Jørgensen, H. D., Johnsen, S. G. and Krogstie, J. *Process Models as a Framework for Knowledge Sharing and Reuse in Extended Enterprises*, IKNOW Conference, Graz, Austria, 2002.
374. Oldevik, J. *SEASPRITE Project homepage*, SINTEF, Oslo, Norway, 1999, *http://www.informatics.sintef.no/DistributedInformationSystems/projects/seasprite.html*.
375. Oldevik, J., Solberg, A., Elvesæter, B. and Berre, A. J. *Framework for Model transformation and Code Generation*, Enterprise Distributed Object Computing (EDOC) Conference, 2002.
376. Olsen, G. *The Emperor has no Lab Coat*, ACM Interactions, vol. 9, no. 4, 2002.
377. OMG *Meta-Object Facility (MOF) Specification v 1.3*, OMG - Object Management Group, 2000.
378. OMG *Workflow Management Facility v. 1.2*, Object Management Group, 2000.
379. OMG *Enterprise Distributed Object Computing (EDOC) Process Profile*, Object Management Group, 2001.
380. OMG *Software Process Engineering Metamodel (SPEM)*, Object Management Group, 2001.
381. OMG *UML Specification v. 1.4*, Object Management Group, 2001.
382. OMG *UML Specification v. 1.5*, Object Management Group, 2002.
383. OMG and 3C *Unified Modeling Language 2.0 Infrastructure Proposal (revised submission), Version 2.0.13*, Object Management Group, 2002.
384. Opdahl, A. L. *Performance Engineering during Information Systems Development*, PhD-thesis, Norwegian Institute of Technology, Trondheim, Norway, 1992.
385. Opdahl, A. L. and Sindre, G. *Facet Modelling: An Approach to Flexible and Integrated Conceptual Modelling*, Information Systems, vol. 22, no. 5, 1997.
386. Orlikowski, W. J. *CASE Tools as Organizational Change: Investigating Incremental and Radical Changes in Systems Development*, MIS Quarterly, vol. 17, no. 3, 1993.
387. Orr, J. *Talking about Machines*. Cornell University Press, Ithaca, New York, 1996.

388. Ossher, H. and Tarr, P. *Using Multi-Dimensional Separation of Concerns to (Re)Shape Evolving Software*, Communications of the ACM, vol. 44, no. 10, 2001.
389. Osterweil, L. J. *Software Processes are Software too,* ICSE Conference, 1987.
390. Ouksel, A. M. and Watson, J. *The Need for Adaptive Workflow and What is Currently Available on the Market,* ACM CSCW Workshop: Towards Adaptive Workflow Systems, Seattle, 1998.
391. Ould, M. A. *Business Processes - Modeling and Analysis for Re-engineering and Improvement*. John Wiley & Sons, Beverly Hills, 1995.
392. Parnas, D. L. *On the Criteria to be Used in Decomposing Systems into Modules,* Communications of the ACM, vol. 15, no. 12, 1972.
393. Parsons, J. *An Information Model Based on Classification Theory,* Management Science, vol. 42, no. 10, 1996.
394. Parsons, J. and Wand, Y. *Choosing Classes in Conceptual Modeling,* Communications of the ACM, vol. 40, no. 6, 1997.
395. Parsons, J. and Wand, Y. *Using Objects for Systems Analysis,* Communications of the ACM, vol. 40, no. 12, 1997.
396. Parsons, J. and Wand, Y. *Emancipating Instances from the Tyranny of Classes in Information Modeling*, ACM Transactions on Database Systems, vol. 25, no. 2, 2000.
397. Perry, J. *Davidson's Sentence Holism and Wittgenstein's Builders,* Proceedings and Addresses of the American Philosophical Association, vol. 68, no. 2, 1994.
398. Petroski, H. *Invention by Design*. Harvard University Press, Cambridge, USA, 1996.
399. Pinkwart, N., Hoppe, U. and Grasser, K. *Integration of Domain-Specific Elements into Visual Language Based Collaborative Environments*, CRIWG Workshop, Darmstadt, Germany, 2001.
400. PMBOK *The Project Management Book of Knoweldge.* Project Management Institute, Newtown Square, Pennsylvania, 2000.
401. Pohl, K., Dömges, R. and Jarke, M. *Decision Oriented Process Modelling,* International Software Process Workshop, 1994.
402. Polanyi, M. *The tacit dimension*. Routledge and Kegan Paul, London, 1966.
403. Pour, G., Glass, M. L. and Lutz, M. *The Push to Make Software Engineering Respectable*, IEEE Computer, vol. 33, no. 5, 2000.
404. Qureshi, S. and Zigurs, I. *Paradoxes and Prerogatives in Global Virtual Collaboration,* Communications of the ACM, vol. 44, no. 12, 2001.
405. Ramduny, D., Dix, A. and Rodden, T. *Exploring the Design Space for Notification Servers,* ACM CSCW Conference, Seattle, USA, 1998.
406. Ramesh, B. *Process Knowledge Management with Traceability*, IEEE Software, vol. 19, no. 3, 2002.
407. Randall, D. and Rouncefield, M. *The Theory and Practice of Fieldwork for Systems Development, Tutorial Notes,* ACM CSCW Conference, Seattle, USA, 1998.
408. Reichert, M., Bauer, T. and Dadam, P. *Enterprise-Wide and Cross-Enterprise Workflow-Management: Challenges and Research Issues for Adaptive Workflows*, Enterprise-Wide and Cross-Enterprise Workflow Management Workshop, Paderborn, Germany, 1999.
409. Ribo, J. M. and Franch, X. *Building Expressive and Flexible Process Models Using a UML-Based Approach*, EWSPT Conference, Springer LNCS 2077, 2001.
410. Ricœur, P. *Metaphore et reference,* in *La methaphore vive*. du Seuil (Norwegian edition "Eksistens og hermeneutikk", Aschehoug, 1999), Paris, France, 1975.
411. Riehle, D., Fraleigh, S., Bucha-Lassen, D. and Omorogbe, N. *The Architecture of a UML Virtual Machine*, ACM OOPSLA Conference, Tampa, USA, 2001.
412. Roberts, D. and Hanmer, R. *Pattern Languages of Programming (PLoP) homepage,* http://jerry.cs.uiuc.edu/~plop/, 1997.
413. Robillard, P. N. *The Role of Knowledge in Software Development*, Communications of the ACM, vol. 42, no. 1, 1999.
414. Robinson, M. *Computer Supported Co-operative Work: Cases and Concepts*, Groupware Conference, Utrecht, Netherlands, 1991.
415. Rodden, T. *Populating the Application: A Model of Awareness for Cooperative Applications*, ACM CSCW Conference, Boston, USA, 1996.
416. Rolfsen, R. K. *Distribuert Samarbeid- Hva skjer? (Distributed Work - What's happening?),* M.Sc.-thesis, Department of Informatics, University of Oslo, Norway, 1997.
417. Rombach, H. D. and Verlage, M. *Directions in Software Process Research,* Advances in Computers, vol. 41, 1995.

418. Rose, T. *Visual Assessment of Engineering Processes in Virtual Enterprises*, Communications of the ACM, vol. 41, no. 12, 1998.
419. Ruggles, R. L. *Knowledge Management Tools*. Butterworth-Heinemann, Boston, USA, 1997.
420. Rumbaugh, J., Jacobson, I. and Booch, G. *The Unified Modeling Language Reference Manual*. Addison Wesley, Reading, Massachusetts, 1999.
421. Runde, R. K. and Stølen, K. *What is Model Driven Architecture?*, Research Report 304, Department of Informatics, University of Oslo, Norway, 2003.
422. Rupprecht, C., Fünffinger, M., Knublauch, H. and Rose, T. *Capture and Dissemination of Experience about the Construction of Engineering Processes*, CAiSE Conference, Springer LNCS 1789, Stockholm, Sweden, 2000.
423. Rupprecht, C., Rose, T., Halm, E. v. and Zwegers, A. *Project-Specific Process Configuration in Virtual Enterprises*, Design of Information Infrastructure Systems for Manufacturing (DIISM) Conference, Melbourne, Australia, 2000.
424. Rus, I. and Lindvall, M. *Knowledge Management in Software Enginering - Special Issue Introduction*, IEEE Software, vol. 19, no. 3, 2002.
425. Röpnack, A., Schindler, M. and Schwan, T. *Concepts of the Enterprise Knowledge Medium*, Practical Aspects of Knowledge Management (PAKM) Conference, Basel, Switzerland, 1998.
426. Røyrvik, E. *EXTERNAL Learning History*, The EXTERNAL Project, Trondheim, Norway, 2000.
427. Sachs, P. *Transforming Work: Collaboration, Learning and Design*, Communications of the ACM, vol. 38, no. 9, 1995.
428. Sandor, O., Bogdan, C. and Bowers, J. *Aether: An Awareness Engine for CSCW*, ECSCW Conference, Lancaster, UK, 1997.
429. Sarin, S. *Flexible Workflow Architectures: The New Adaptive Workflow*, Workflow Conference, San-Jose, USA, 1995.
430. Sarin, S. K., Abbott, K. R. and McCarthy, D. R. *A Process Model and System for Supporting Collaborative Work*, ACM Conference on Organizational Computing Systems (COOCS), 1991.
431. Scagno, G. *Evaluation of the EXTERNAL Tools Applied to EXTERNAL Use Case*, Deliverable from the EXTERNAL Project, Patras, Greece, 2002.
432. Scheer, A.-W. and Haberman, F. *Making ERP a Success*, Communications of the ACM, vol. 43, no. 4, 2000.
433. Scheer, A.-W. and Hoffmann, M. *From Business Process Model to Application System - Developing an Information System with the House of Business Engineering (HOBE)*, CAiSE Conference, Springer LNCS 1626, 1999.
434. Scheer, A.-W. and Nuttgens, M. *ARIS Architecture and Reference Models for Business Process Management*, in *Business Process Management*, W. v. d. Aalst, J. Desel, and A. Oberweis, Eds. Springer LNCS 1806, Berlin, Germany, 2000.
435. Schleicher, A., Westfechtel, B. and Jäger, D. *Modeling Dynamic Software Processes with UML*, Technical Report AIB 98-11, RWTH, Aachen, Germany, 1998.
436. Schlichter, J., Koch, M. and Bürger, M. *Workspace Awareness for Distributed Teams*, Coordination Technology for Collaborative Applications Workshop, Singapore, 1997.
437. Schmidt, K. *Of maps and scripts - The status of formal constructs in cooperative work*, ACM GROUP Conference, Phoenix, Arizona USA, 1997.
438. Schmidt, K. and Bannon, L. *Taking CSCW Seriously - Supporting Articulation Work*, Computer Supported Cooperative Work, vol. 1, no. 1, 1992.
439. Schmidt, K. and Simone, C. *Coordination Mechanisms: Towards a Conceptual Foundation of CSCW Systems Design*, Computer Supported Cooperative Work, vol. 5, 1996.
440. Schmidt, M.-T. *Building Workflow Business Objects*, OPPSLA Business Object Workshop, Vancouver, Canada, 1998.
441. Schneider, K. *What to Expect from Software Experience Exploitation*, IKNOW Conference, Graz, Austria, 2002.
442. Schuette, R. *Architecture for Evaluating the Quality of Information Models - A Meta and Object Level Comparison*, ER Conference, Paris, France, 1999.
443. Schuler, C., Schuldt, H., Alonso, G. and Schek, H.-J. *Workflows over Workflows: Practical Experience with the Integration of SAP R/3 Business Workflows in WISE*, Informatik Workshop, Paderborn, Germany, 1999.
444. Schön, D. A. *Organizational Learning*, in *Beyond Method*, G. Morgan, Ed. Sage, Beverly Hills, 1983.

445. Schön, D. A. *The Reflective Practitioner*. Ashgate Publishing Ltd., Aldershot, UK, 1983.
446. Seltveit, A. H. *Complexity Reduction in Information Systems Modelling*, PhD-thesis, The Norwegian Institute of Technology, Trondheim, Norway, 1994.
447. Senge, P. *The Fifth Discipline: The Art and Practice of the Learning Organization*. Century Business Publishers, London, 1990.
448. Shapiro, D. *The Limits of Ethnography: Combining Social Sciences for CSCW*, ACM CSCW Conference, Chapel Hill, NC, USA, 1994.
449. Sharp, H., Robinson, H. and Woodman, M. *Software Engineering: Community and Culture*, IEEE Software, vol. 17, no. 1, 2000.
450. Shim, S. S. Y., Pendyala, V. S., Sundaram, M. and Gao, J. Z. *Business-to-Business E-Commerce Frameworks*, IEEE Computer, vol. 33, no. 10, 2000.
451. Shipman, F., Airhart, R., Hsieh, H., Maloor, P., Moore, J. M. and Shah, D. *Visual and Spatial Communication and Task Organization Using the Visual Knowledge Builder*, ACM GROUP Conference, Boulder, USA, 2001.
452. Shipman, F. M. and Marshall, C. C. *Formality Considered Harmful: Experiences, Emerging Themes, and Directions of Use of Formal Representations in Interactive Systems*, Computer Supported Cooperative Work, vol. 8, no. 4, 1999.
453. Shipman, F. M. and McCall, R. J. *Supporting knowledge-base evolution with incremental formalization*, ACM Human Factors in Computing Systems Conference, Boston, MA, 1994.
454. Shipman, F. M. and McCall, R. J. *Incremental formalization with the hyper-object substrate*, ACM Transactions on Information Systems, vol. 17, no. 2, 1999.
455. Shum, S. B. *Representing Hard-to-Fomalise, Contextualised, Multidisciplinary, Organisational Knowledge*, AAAI Spring Symposium on Artificial Intelligence in Knowledge Management, Stanford University, Palo Alto, USA, 1997.
456. Simone, C., Divitini, M. and Schmidt, K. *A notation for malleable and interoperable coordination mechanisms for CSCW systems*, ACM Conference on Organizational Computing Systems (COOCS), Milpitas, CA, USA, 1995.
457. Singh, B. and Rein, G. L. *Role Interaction Nets (RINs); A Process Description Formalism*, Technical Report CT-083-92, MCC, Austin, Texas, 1992.
458. Singh, M. P. *Conceptual Modeling for Multiagent Systems: Applying Interaction-Oriented Programming*, in *Conceptual Modelling*, P. P. Chen, J. Akoka, H. Kangassalo, and B. Thalheim, Eds. Springer LNCS 1565, Berlin, Germany, 1999.
459. Smith, H. *A System Integrator's Perspective on Business Process Management, Workflow and EAI*, Infoconomy Agile Business Conference, 2002.
460. Soh, C., Kien, S. and Tay-Yap, J. *Cultural Fits and Misfits: Is ERP a Universal Solution?*, Communications of the ACM, vol. 43, no. 4, 2000.
461. Sorenson, P. G., Findeisen, P. S. and Tremblay, J. P. *Supporting Viewpoints in Metaview*, ACM SIGSOFT Workshop, San Fransisco, CA, USA, 1996.
462. Star, S. L. and Griesemer, J. R. *Institutional Ecology, 'Translations' and Boundary Objects: Amateurs and Professionals in Berkeley's Museum of Vertebrate Zoology, 1907-39*, Social Studies of Science, vol. 19, 1989.
463. Stewart, N. F. *Science and Computer Science*, ACM Computing Surveys, vol. 27, no. 1, 1995.
464. Strong, D. M. and Miller, S. M. *Exceptions and Exception Handling in Computerized Information Processes*, ACM Transactions on Information Systems, vol. 13, no. 2, 1995.
465. Strømseng, K., Olsson, N., Haake, J. and Scagno, G. *EE Requirements*, Deliverable 3-31-D-2000-01-1, The EXTERNAL Project, Oslo, Norway, 2000.
466. Styhre, A. *Thinking with AND: Management Concepts and Multiplicities*, Organization, vol. 9, no. 3, 2002.
467. Störle, H. *Describing Process Patterns with UML*, EWSPT 2001, Springer LNCS 2077, 2001.
468. Suchman, L. *Plans and Situated Actions*. Cambridge University Press, New York, 1987.
469. Suchman, L. *Do Categories Have Politics?*, Computer Supported Cooperative Work, vol. 2, no. 3, 1994.
470. Suchman, L. *Making Work Visible*, Communications of the ACM, vol. 38, no. 9, 1995.
471. Suchman, L. *Representation of Work - Introduction to Special Issue*, Communications of the ACM, vol. 38, no. 9, 1995.
472. Suchman, L. *Speech Acts and Voices: Response to Winograd et al.*, Computer Supported Cooperative Work, vol. 3, no. 1, 1995.

473. Sullivan, G. T. *Aspect-Oriented Programming Using Reflection and Metaobject Protocols*, Communications of the ACM, vol. 44, no. 10, 2001.
474. Sun Microsystems *Java Servlets*, *http://java.sun.com/products/servlet/*, 1996.
475. Swenson, K. *SWAP - Simple Workflow Access Protocol*, IETF, 1998.
476. Swenson, K. *Workflow for the Information Worker*, in *Workflow Handbook 2001*, L. Fischer, Ed. Workflow Management Coalition, Future Strategies Inc., Lighthouse Point, Florida, USA, 2000.
477. Swenson, K. D., Irwin, K., Matsumoto, T., Maxwel, R. J. and Saghari, B. *Collaborative Planning: Empowering the User in a Process Support Environment*, Interdisciplinary Workshop on Informatics and Psychology, Schaerding, Austria, 1994.
478. Swenson, K. D., Maxwell, R. J., Matsumoto, T., Saghari, B. and Irwin, I. *A Business Process Environment Supporting Collaborative Planning*, Journal of Collaborative Computing, vol. 1, no. 1, 1994.
479. Söderström, E., Andersson, B., Johannesson, P., Perjons, E. and Wangler, B. *Towards a Framework for Comparing Process Modelling Languages*, CAiSE Conference, Springer LNCS 2348, Toronto, Canada, 2002.
480. Sølvberg, A. *Data and What They Refer to*, in *Conceptual Modeling*, P. P. Chen, J. Akoka, H. Kangassalo, and B. Thalheim, Eds. Springer LNCS 1565, Berlin, Germany, 1999.
481. Sølvberg, A. and Brasethvik, T. *The Referent Model Language*, Technical Report, Norwegian University of Science and Technology, Department of Computer and Information Science, 1997.
482. Sølvberg, A. and Kung, D. C. *Information Systems Engineering - An Introduction*. Springer, Berlin, Germany, 1993.
483. Taivalsaari, A. *On the Notion of Inheritance*, ACM Computing Surveys, vol. 28, no. 3, 1996.
484. Tang, J. and Hwang, S.-Y. *Handling Uncertainties in Workflow Applications*, ACM CSCW Conference, Boston, USA, 1996.
485. Taylor, F. W. *The Principles of Scientific Management*. Dover Publications, Inc. (this edition 1998), Mineola, New York, USA, 1911.
486. Teege, G. *HieraStates: Flexible Interaction with Objects*, Report TUM-I9441, Technische Universität München, 1994.
487. Teege, G. *HieraStates: Supporting Workflows which Include Schematic and Ad-Hoc Aspects*, Practical Aspects of Knowledge Management (PAKM) Conference, Basel, Switzerland, 1996.
488. Teege, G. *Users as Composers: Parts and Features as a Basis for Tailorability in CSCW Systems*, Computer Supported Cooperative Work, vol. 9, no. 1, 2000.
489. Telesius, E. and Jaliniauskas, A. *The Analysis and Implementation of Ad-hoc Business Processes Based on Common Approximation Patterns*, Workflow Management Conference, Münster, Germany, 1999.
490. Tichy, W. F. *Should Computer Scientists Experiment More?*, IEEE Computer, vol. 31, no. 5, 1998.
491. Tinella, S., Karlsen, D., Lillehagen, F. and Smith, M. W. *Model-driven operational solution*, Concurrent Engineering (CE) Conference, Madeira, Portugal, 2003.
492. Tolvanen, J.-P. and Lyytinen, K. *Flexible Method Adaptation in CASE: The Metamodelling Approach*, Scandinavian Journal of Information systems, vol. 5, 1993.
493. Trigg, R. H. and Bødker, S. *From Implementation to Design: Tailoring and the Emergence of Systematization in CSCW*, ACM CSCW Conference, Chapel Hill, North Carolina, USA, 1994.
494. Tripp, L. L. *Benefits of Certification*, IEEE Computer, vol. 35, no. 6, 2002.
495. Trist, E. L. *The Evolution of Socio-Technical Systems*, Issues in the Quality of Working Life, no. 2, 1981.
496. Trist, E. L. and Bamforth, K. W. *Some Social and Psychological Consequences of the Longwall Method of Coal-Getting*, Human Relations, vol. 4, no. 1, 1951.
497. Truex, D. P., Baskerville, R. and Klein, H. *Growing Systems in Emergent Organizations*, Communications of the ACM, vol. 42, no. 8, 1999.
498. Trætteberg, H. *Model-based User Interface Design*, PhD-thesis, Norwegian University of Science and Technology, Trondheim, Norway, 2001.
499. Twidale, M. B. and Marty, P. F. *Coping with Errors: The Importance of Process Data in Robust Sociotechnical Systems*, ACM CSCW Conference, Philadelphia, USA, 2000.
500. UN/CEFACT and OASIS *ebXML Business Process and Business Information Analysis Overview v. 1.0*, 2001.
501. UN/CEFACT and OASIS *ebXML Business Process Specification Schema v. 1.01*, 2001.

502. Uschold, M. and Gruninger, M. *ONTOLOGIES: Principles, Methods and Applications*, Knowledge Engineering Review, vol. 11, no. 2, 1996.
503. Utterback, J. *Mastering the Dynamics of Innovation*. Harvard Business School Press, Boston, USA, 1994.
504. Vanhoenacker, J., Bryant, A. and Dedene, G. *Creating a Knowledge Management Architecture for Business Process Change*, ACM SIGCPR Conference, New Orleans, USA, 1999.
505. Vetland *Measurement-Based Composite Computational Work Modelling of Software*, PhD-thesis, Norwegian Institute of Technology, Trondheim, Norway, 1993.
506. Voas, J. *Software Quality's Eight Greatest Myths*, IEEE Software, vol. 16, no. 5, 1999.
507. Voorhoeve, M. *Compositional Modelling and Verification of Workflow Processes*, in *Business Process Management*, W. v. d. Aalst, J. Desel, and A. Oberweis, Eds. Springer LNCS 1806, Berlin, Germany, 2000.
508. Voss, A., Procter, R., Herrmann, T. and Jørgensen, H. D. *Structure and Process: The Interplay of Routine and Informed Action*, ECSCW 2001 Workshop, Bonn, Germany, 2001.
509. Voss, A., Procter, R. and Williams, R. *Innovation in Use: Interleaving day-to-day operation and systems development*, Participatory Design Conference, New York, USA, 2000.
510. Välikangas, L. *Corporate Renewal in a Knowledge Network*, Report R843, SRI Consulting, Business Intelligence Program, Menlo Park, CA, USA, 1997.
511. Wactlar, H. D. *Extracting and Visualizing Knowledge from Film and Video Archives*, IKNOW Conference, Graz, Austria, 2002.
512. Wand, Y. and Weber, R. *On the Deep Structure of Information Systems*, International Conference on Information Systems, Copenhagen, Denmark, 1990.
513. Wang, A. I. *Using a Mobile, Agent-based Environment to Support Cooperative Software Processes*, PhD-thesis, Norwegian University of Science and Technology, Trondheim, Norway, 2001.
514. Wang, W.-L. *Beware of the Engineering Metaphor*, Communications of the ACM, vol. 45, no. 6, 2002.
515. Warboys, B. C., Balasubramaniam, D., Greenwood, R. M., Kirby, G. N. C., Mayes, K., Morrison, R. and Munro, D. S. *Instances and Connectors: Issues for a Second Generation Process Language*, EWSPT Conference, Springer LNCS 1487, Weybridge, UK, 1998.
516. Warboys, B. C., Balasubramaniam, D., Greenwood, R. M., Kirby, G. N. C., Mayes, K., Morrison, R. and Munro, D. S. *Collaboration and Composition: Issues for a Second Generation Process Language*, European Software Engineering Conference (ESEC), Springer LNCS 1687, Toulouse, France, 1999.
517. Warmer, J. and Kleppe, A. *The Object Constraint Language*. Addison-Wesley, 1999.
518. Wasserschaff, M. and Bentley, R. *Supporting Cooperation through Customisation: The Tviews Approach*, Computer Supported Cooperative Work, vol. 6, 1997.
519. Weber, M. *Makt og Byråkrati* (Norwegian edition, Gyldendal, Oslo, 1971), 1922.
520. Webster *The New International Webster's Comprehensive Dictionary of the English Language*. Trident Press, 1996.
521. Wegner, P. *Research Paradigms in Computer Science*, ICSE Conference, San Francisco, CA, USA, 1976.
522. Wegner, P. *Interaction as a basis for empirical computer science*, ACM Computing Surveys, vol. 27, no. 1, 1995.
523. Wegner, P. *Why interaction is more powerful than algorithms*, Communications of the ACM, vol. 40, no. 5, 1997.
524. Wegner, P. and Goldin, D. *Interaction as a Framework for Modeling*, in *Conceptual Modeling*, P. P. Chen, J. Akoka, H. Kangassalo, and B. Thalheim, Eds. Springer LNCS 1565, Berlin, Germany, 1999.
525. Wegner, P. and Goldin, D. *Mathematical Models of Interactive Computing*, Brown Technical Report CS 99-13, Computer Science Department, Brown University, Providence, RI, USA, 1999.
526. Weingarten, R. *Government Funding and Computing Research Priorities*, ACM Computing Surveys, vol. 27, no. 1, 1995.
527. Wenger, E. *Communities of Practice. Learning Meaning and Identity*. Cambridge University Press, Cambridge, UK, 1998.
528. Weske, M., Goesmann, T., Holten, R. and Striemer, R. *A Reference Model for Workflow Application Development Processes*, ACM Work Activities Coordination and Collaboration Conference (WACC), San Francisco, USA, 1999.

529. WfMC *The Workflow Reference Model, Version 1.1*, Standard TC00-1003, Workflow Management Coalition, 1994.
530. WfMC *Workflow Management Coalition, Interface 1: Process Definition Interchange. Draft 5.0, Issued on February 03, 1996*, Workflow Management Coalition, 1996.
531. WfMC *Terminology & Glossary*, Standard TC-1011, Issue 3.0, Workflow Management Coalition, 1999.
532. WfMC *Workflow Handbook 2001*. Workflow Management Coalition, Future Strategies Inc., Lighthouse Point, Florida, USA, 2000.
533. Wikarski, D. *An Introduction to Modular Process Nets*, Technical Report 96-019, International Computer Science Institute, Berkeley, CA, 1996.
534. Williams, L., Kessler, R. R., Cunningham, W. and Jeffries, B. *Strengthening the Case for Pair Programming*, IEEE Software, vol. 17, no. 4, 2000.
535. Willumsen, G. *Executable Conceptual Models in Information Systems Engineering*, PhD-thesis, Norwegian Institute of Technology, Trondheim, Norway, 1993.
536. Winograd, T. *A Language/Action Perspective on the Design of Cooperative Work*, Human-Computer Interaction, vol. 3, 1987.
537. Winograd, T. *Categories, Disciplines and Social Coordination*, Computer Supported Cooperative Work, vol. 2, no. 3, 1994.
538. Winograd, T. and Flores, F. *Understanding Computers and Cognition*. Addison-Wesley, 1986.
539. Wulf, W. A. *Are We Scientists or Engineers?*, ACM Computing Surveys, vol. 27, no. 1, 1995.
540. Wyner, G. M. and Lee, J. *Applying Specialization to Process Models*, ACM Conference on Organizational Computing Systems (COOCS), Milplitas, CA, USA, 1995.
541. Yang, G. *Towards a Library for Process Programming*, in *Business Process Management*. Springer LNCS 2678, Berlin, Germany, 2003.
542. Young, P. S. C. *Customizable Process Specification and Enactment for Technical and Non-Technical Users*, PhD-thesis, University of California, Irvine, USA, 1994.
543. Yu, E. S. K. and Mylopoulos, J. *Using Goals, Rules, and Methods to Support Reasoning in Business Process Reengineering*, Hawaii International Conference on Systems Sciences (HICCS'27), Maui, Hawaii, US, 1994.
544. Zanchi, M., Su, X. and Gulla, J. A. *Modelling with APM in ERP projects*, Open Enterprise Solutions: Systems, Experiences, and Organizations Conference, Rome, Italy, 2001.
545. Zelkowitz, M. V. and Wallace, D. R. *Experimental Models for Validating Technology*, IEEE Computer, vol. 31, no. 5, 1998.
546. Zelm, M. *Towards User Oriented Enterprise Modelling - Comparison of Modelling Languages*, Concurrent Engineering (CE) Conference, Madeira, Portugal, 2003.
547. Zuboff, S. *In the Age of the Smart Machine*. Basic Books Inc, USA, 1988.
548. Aalst, W. M. P. v. d. *The Application of Petri Nets to Workflow Management*, The Journal of Circuits, Systems and Computers, vol. 8, no. 1, 1998.
549. Aalst, W. M. P. v. d. *Formalization and Verification of Event-driven Process Chains*, Information and Software Technology, vol. 41, no. 10, 1999.
550. Aalst, W. M. P. v. d. and Basten, T. *Life-cycle Inheritance: A Petri-net-based approach*, in *Application and Theory of Petri Nets*, P. Azema and G. Balbo, Eds. Springer LNCS 1248, Berlin, Germany, 1997.
551. Aalst, W. M. P. v. d. and Berens, P. J. S. *Beyond Workflow Management: Product-Driven Case Handling*, ACM GROUP Conference, Boulder, USA, 2001.
552. Aalst, W. M. P. v. d., Hofstede, A. H. M. t., Kiepuszewski, B. and Barros, A. P. *Workflow Patterns*, Distributed and Parallel Databases, 2003.
553. Aalst, W. v. d., Desel, J. and Oberweis, A. *Business Process Management*. Springer LNCS 1806, Berlin, Germany, 2000.
554. Aalst, W. v. d., Moldt, D., Valk, R. and Wienberg, F. *Enacting Interorganizational Workflows Using Nets in Nets*, Workflow Management Conference, Münster, Germany, 1999.