Darijus Strašunskas

Domain Model-Centric Distributed Development

An approach to semantics-based change impact management

Doctoral thesis
for the degree of doktor ingeniør

Trondheim, 2006

Norwegian University of Science and Technology
Faculty of Information Technology, Mathematics
and Electrical Engineering
Department of Computer and Information Science

NTNU
Innovation and Creativity

NTNU
Norwegian University of Science and Technology
Doctoral thesis
for the degree of doktor ingeniør
Faculty of Information Technology, Mathematics
and Electrical Engineering
Department of Computer and Information Science

ISBN 82-471-7762-5 (printed ver.)
ISBN 82-471-7761-7 (electronic ver.)
ISSN 1503-8181

Doctoral theses at NTNU, 2006:12

Printed by NTNU-trykk

To my family
for their love and patience

Abstract

Today's information systems engineering involves large number of stakeholders, wide geographical distribution and wide range of tools. Success in system engineering depends on effective human communication. Early understanding and modelling of the problem domain is a key to manage large scale systems and projects. This requires stakeholders to reach a certain level of shared interpretation of the domain referred throughout the development

We propose a method for semantics driven change impact assessment. In our method, first a collaborative problem analysis is conducted. The problem analysis results in an agreed and committed common understanding of the problem domain, expressed in a conceptual domain model. The constructed conceptual domain-specific model is then actively used as a communication medium, e.g., to abstract development objects from representation format in order to explicate their semantics. Stakeholders browse the domain model and interactively associate to product fragments by selecting concept clusters that best describe the contents (intended meaning) of the product fragments.

Associations of the development objects with concepts from domain model, as well as the domain model itself constitute the basis for change impact assessment throughout the development. Every revision of a development object invokes change impact notifications that are either confirmed or rejected. Accumulated statistics are used to refine associations via the domain model to the direct dependency links among development objects.

The method has been implemented in a prototype system CO_2SY and has been evaluated in an experiment, where a set of test users has been provided with a problem domain description including a domain model and a set of development objects. The experiment was based on two real world cases. Users were asked to perform tasks using the prototype and two comparative tools. The method and prototype have been evaluated with respect to actual performance and users perceptions. The result shows actual effectiveness, perceived ease of use and usefulness comparing to other tools used in the experiment, as well as intention of the subjects to use the method in future.

A discussion of future research directions and possible revisions of the method concludes the thesis.

Table of Contents

List of Figures

List of Tables

Preface

This thesis is submitted to the Norwegian University of Science and Technology (NTNU) in partial fulfilment of the requirements for the doctoral degree *doktor ingeniør*. The work has been carried out at the Information Systems Group, within the Department of Computer and Information Science (IDI), under supervision of Professor Arne Sølvberg.

Acknowledgment

I would like to acknowledge the effort of my principal advisor Prof. Arne Sølvberg. Arne has given me good guidance when exploring different research directions, has assigned interesting technical and scientific tasks, not necessarily leading to a degree, but still important for my professional development.

I would like to express special thanks to Sari Hakkarainen for constructive discussions when writing my first conference papers, and continuous support through the later phases of the PhD studies, especially, for the proof-reading parts of this thesis. It is a pleasure to record my gratitude for all these discussions on and off the topic we have had in the recent years.

Many people have influenced my work conducted during the studies. In particular I am thankful to 林云 (Yun Lin) for the sincere personal communication and the productive and valuable collaboration that resulted in several co-authored articles. I also would like to thank Assoc. Prof. Rimantas Butleris for introducing me to the field of information systems, for inspiration to begin PhD studies, personal and scientific discussions.

Graduate students Kjell O. Erichsen and Stig Lau, whom I have supervised, have helped a lot in refinement of ideas through interesting discussions – I am thankful for that. I thank Arne Dag Fidjestøl for interesting and memorable cooperation and discussions on a modelling environment, as well as technical support.

I thank my colleagues and friends for a dynamic and encouraging environment. Special thanks go to Christian Mönch for exciting discussions about technological issues and life in general. 苏晓萌 (Xiaomeng Su) and Stein Løkke Tomassen have been sharing office with me during different periods of this work. I thank them for being friendly, positive and helpful. I have enjoyed collaboration with Lillian Hella and Raimundas Matulevičius, thanks for enthusiasm and pragmatism you shared. I express my gratitude to 饶京海 (Jinghai Rao), Csaba Veres, and Jennifer Sampson for being more than colleagues and for providing a mental support. I also thank all former and current members of IS-Group for a good working environment.

Thanks to everybody at IDI for a scientific atmosphere. Special thanks go to administrative and technical staff for providing a necessary infrastructure and helping to smoothly perform all indispensable routines.

I am grateful to my friends both, here in Norway and abroad for helping me to have more cheerful moments and a social life. I have appreciated those few joyful moments that we have shared during short summer holidays in Lithuania, their attitude of looking always to a bright side of life. In particular, I would like to acknowledge Giedrius Romeika for his contribution in proof-reading the thesis.

I am immensely grateful to my family. I thank my mother for always believing in me, for her support, tolerance, readiness to help. I also thank my father and brother for support and encouragements, intensive communication by sms and e-mail. I would like to express my immeasurable gratitude to my wife Kristina and my son Simonas. I am grateful for their love, patience, kindness and for fulfilling my private life with a joy and energy, when I most needed.

Darijus Strašunskas

Trondheim,
February 1st, 2006

1 Introduction

The research reported in this thesis is conducted in the Information Systems (IS) group at the Department of Computer and Information Science at the Norwegian University of Science and Technology (NTNU). IS group has a strong tradition in developing formalisms and tools for information systems modelling. Within this tradition, cooperation has always been considered as an important part of information systems development (Farshchian, 2001; Solvberg, 2000; Andersen, 1994). The need to support geographically distributed groups of developers has emerged with the wide-spread use of Internet in the last decade. Furthermore, many companies nowadays are spread all over the world, with specialized divisions or subsidiary companies producing software components (product fragments) assembled elsewhere. That has further increased the need to manage and facilitate cooperation by facilitating development coordination and management of product compositions. This thesis is developed in the research tradition of the IS group, and contributes to the IS engineering (ISE) research by suggesting and developing methods supported by tools to facilitate cooperation in distributed project teams in general, and management of product fragments in particular.

1.1 Background

Traditional systems development was centred on small projects and collocated teams using a limited toolset. While today's development organisations meet large number of stakeholders, wide geographical distribution and wide range of tools are used (VA Software, 2004). Furthermore, information systems engineering usually includes enterprise, systems and business process modelling. As output, it provides information analysis, architecture and design. Typically, all these parts should be integral. Thus, the end product of IS engineering is not a homogeneous specification, but rather a collection of correlated product fragments (i.e., requirements specification, design, code, test scenarios, and documentation). The fragments have various perspectives, focusing on different aspects, and expressed in different representation languages.

Management of such product fragments is essential for any logically or geographically distributed large scale project. Distributed development projects have special settings and needs, where attention has to be given to product fragments management because developers are likely to use different representation formats and a variety of tools for the product development.

Furthermore, one key issue in software projects is the two-way communication, where the developers understand their clients and the clients understand the issues the developers present. In order to achieve a sufficiently accurate level of communication, each party has to ensure that the meaning of their utterances – the semantics – is successfully understood by their counterpart (Finkelstein et al., 1991). Hence, many distributed development projects need to interchange, possibly heterogeneous, information by explicitly communicating its semantics.

Models are usually built to share knowledge or definitions with other people, and are especially directed to people that want to share knowledge or define knowledge in cooperation with others. Modelling is seen as "the activity of formally describing some aspects of the physical and social world around us for purposes of understanding and communication" (Mylopoulos, 1992), and is applied in the early phases of information system analysis and design. However, many problems are encountered when building models. It is conceivable that a variety of different versions of models will be used in different stages of the development process; in general, it is difficult to develop a model that can be acceptable for all participants in a development project. Furthermore, different people usually present different models given the same domain and the same problem. The same information about a system may be modelled at various levels of abstraction and from different viewpoints considering different aspects. Variations among models generally appear due to the creative nature of the modelling activity, as well as other factors such as the richness of the modelling language (Moriarty, 2000), the ambiguities of modelling grammars, and others.

1.2 Problem

System development is a complex and difficult task. It is usually a creative and collaborative process, during which different stakeholders are focusing on various aspects, expressing them at different levels of abstraction, and producing several variants of each (Figure 1.1). *Different levels of abstraction* of the same system enable dealing with complexity by removing details from the model. The model must be designated as an efficient and effective communications medium between the different parties involved in a development project. Usually, an intermediate model is augmented by details in each development step, until it contains sufficient details for execution. *Different aspects* concern multiple views, one no more detailed than the other, e.g., different subject areas. A system can be described from many viewpoints. Each viewpoint defines what characteristics should be included in its views and what issues should be ignored or treated as transparent. A viewpoint is, therefore, a piece of the model that is small enough to comprehend but that also contains all relevant information about a particular concern. *Different variants* concern multiple versions and configurations of the pieces of models.

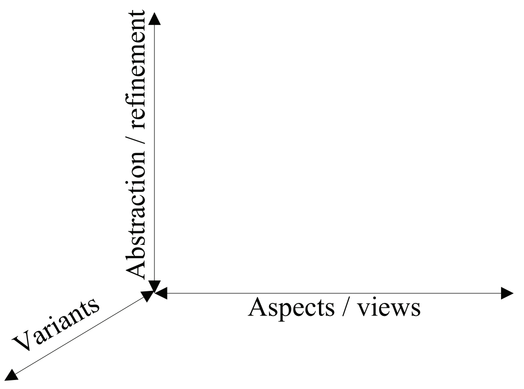


Figure 1.1 Product development dimensions

However, there is big diversity of representation formats and modelling languages that are used throughout the development process. System specifications consist of a wide variety of product fragments (development objects)[1], i.e. different pieces of information about a particular system that together comprise a full (or partial) system specification at various levels of abstraction. Some of these product fragments are well structured, like textual or graphical documents, while others are more loosely structured; therefore, traversing the growing specification between different product fragments is not trivial.

Tasks in the distributed projects are assigned based on the competence of the involved parties. Therefore, developers may use different tools to create and modify product fragments. Furthermore, the fragments can be refined iteratively and be further interchanged among members of a project. It is important for colleagues to interpret a piece of specification correctly. More precisely, Farshchian (2001) emphasized a list of requirements for product development environments to enable collaboration in geographically distributed developments. There we adopt the requirements as follows.

Flexible access to the product. A product development environment should provide flexible mechanisms for accessing and updating the product.

[1] The notion of *product fragment* and *development object* is used interchangeably in this thesis, depending on the context.

Unrestricted product object types. A product development environment should allow the developers to share any type of object that they might find useful for supporting their cooperation.

Unrestricted relation types. A product development environment should allow the developers to create any type of relation between any two objects of product.

Incremental product refinement. A product development environment should provide the developers with flexible mechanisms for incrementally refining the product. The developers should be allowed to start with vague products, and to refine them into more complete and formal ones.

Support for boundary objects. A product development environment should allow the developers to view the product from different perspectives. The environment should in addition support a global view of the product.

Active delivery of information. A product development environment should take an active part in delivering necessary information to the developers. In particular information about changes to the shared product should be delivered continuously to the interested developers.

These are the main challenges - to interrelate and trace all fragments of system specification in different representation formats that are produced in a distributed manner using different tools throughout the whole product lifecycle, i.e., "having divided to conquer, we must now reunite to rule" (Jackson, 1990).

Thus, there is a need to support information systems and service development, which could be achieved by providing means for semantic interoperability and management of specification fragments independently of development phase, model perspective, view or representation language. The need for such approach stems from the fact that in a distributed project development process different tools and, most likely, different notations will be used during the project. The main problem is exchanging heterogeneous information with explicitly communicated meaning. How to relate all pieces of information produced in such a project is the research question we are pursuing.

1.3 Objective

The overall objective of this work is to introduce a method for distributed collaborative work environment supporting management and change impact prediction of the diverse product fragments based on the semantics of the product fragments. This objective is decomposed into four core intermediate research goals determining the development of this work. They are as follows.

- To investigate what means can support cooperative distributed development by providing a common reference space to represent the semantics of development objects;

- To explore how developers can commit to and use that common reference point throughout the whole development lifecycle;
- To elaborate how semantics of the development objects can benefit distributed system development by facilitating change impact assessment;

and then:

- To investigate whether observed change impact notifications can be useful to establish direct dependency links between development objects.

1.4 Approach

The main intention is to investigate how heterogeneous information can be explicated in distributed project. Here we propose to tackle the challenge of describing heterogeneous information by semantically enriching the product fragments, i.e. providing means to explicate their meaning. An important aspect here is that the developers most likely are going to use different object formats, e.g., different modelling languages to specify their own views. Even when using a language with explicitly defined semantics, the meaning of a constituent piece of product can be difficult to interpret to one who is not familiar with that particular language. Semantic enrichment facilitates product management by explicating different kinds of 'hidden' information concerning semantics of the created objects.

> *"Data semantics is the relationship between data and what the data stand for. In order to obtain mutual understanding of interchanged data, the actors have to share a model of what the data represent. Semantic interoperability is about how to achieve such mutual understanding."* Solvberg et al. (2002)

The approach is twofold: First, provide means for distributed collaborative modelling in order to produce a shared conceptual domain model. Second, manage the IS engineering process as well as the product under development by means of a common conceptualisation of the domain.

1.4.1 Common conceptualisation of domain

The approach relies on adoption of a conceptual domain model by all stakeholders, which implicates that IS developers should agree and share conceptualisation of problem. That is important for feasibility of the product fragments management part of the approach.

The meaningfulness of the shared conceptualisation is dependent on how their representation became a common one. Although, the initial goal is usually to develop a single model of the UoD (Universe of Discourse), it often turns out to be important to preserve and model the various "views" of the information as seen by different stakeholders and participants during the system analysis phase. Usually, different developers might have different vocabulary to express their perception of the world. It is important to preserve the knowledge as possessed by the developer and expressed in the model fragment she has developed. We need to ensure that a developer's work will not be disturbed. For instance, if a developer uses term 'aircraft' referring to 'airplane', this term should be preserved in her local view, otherwise after several changes it will be difficult to continue.

In order to sustain meaningfulness of shared conceptualisation, we endeavour to provide an enhanced modelling environment, where stakeholders can model, share, discuss and agree about their conceptualized views (i.e., model fragments), gradually composing them into a complete model. The collaborative modelling phase results in a pragmatic agreement on conceptualisation.

1.4.2 Product fragment management

The underlying rationale here is the belief that the richer semantic information the product fragment could reveal, the more precise accounts of them could be made and in turn the dependency between the fragments could be discovered.

The enrichment is conducted by associating each development object with the corresponding concepts from both a specific domain model (project-dependent). The facts that concepts are interrelated with each other in the domain model and all fragments are linked to domain concepts enable us to derive semantic relationships between different fragments. The enriched semantic information is added into metadata (information/data about data) to abstract away from heterogeneous representation details and capture information content. The conceptual model is used to interoperate across different representation formats used in system development. In particular, product fragments association with the concepts from a domain model adds on to the semantics of the fragment providing a view of what part of a problem domain the fragment describes, for instance, a purchase order.

In the domain model, all concepts are interrelated with generic hierarchical relationships or specific, weighted relations. Those weights are assigned according to how strongly concepts are related. Meanwhile, the development objects are linked to concepts of the domain model. Relations between fragments and concepts are based on the semantics of the fragments. The hierarchical position of the concept, semantics of the generic relationships between concepts and the specific-weighted relations are used to relate heterogeneous fragments, after having constructed such a model and the links. The model with the links is further used to estimate likelihood of impact from altering one fragment to another. Fragments linking through the domain model (marked by '-!-' in Figure 1.2) are gradually refined to direct (more precise) dependency links between fragments (marked by '-?-' in Figure 1.2). In other words, two kinds of relationships are used for product fragments inter-linking: semantic associations between fragment and concept; and direct relationships between fragments based on dependency between them.

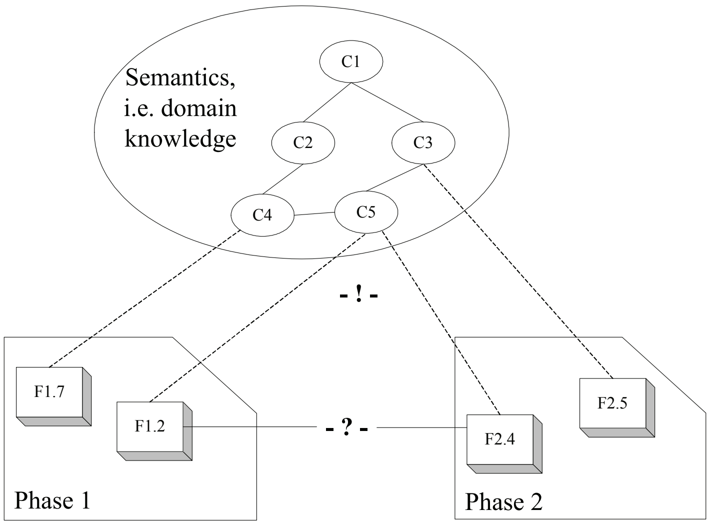


Figure 1.2 Conceptual view of the approach

To sum up, two major techniques constitute the product fragments management approach. First, a modelling framework defining means for externalising stakeholders' knowledge about problem domain, sharing it, internalising and, finally, committing to a shared model. Second, a product fragment management framework providing means to associate development objects with the shared model, in order to enrich their semantics and to enable content change management.

1.5 Main Contributions

The main contributions of this thesis are the development and the specification of the method for product constituent fragments management, impact prediction in geographically distributed systems development. Namely:

- A framework for distributed product fragments management to facilitate the coordination and management of the development process, and methodological approach for integration and manipulation of all information produced during the project;
- A generic implementation of the framework. The specified method resulted in implementation of a repository system.

As a necessary and integral part of the method is development of environment for collaborative modelling as an instrument to achieve common conceptualisation of problem on-hands; and there the contributions are:

- An enhanced method to support collaborative distributed modelling;
- An extended model composition management framework.

Figure 1.3 summarizes goals of this thesis and proposed means to achieve them. Namely, the modelling environment is used for the model fragment management and to reach common conceptualisation of domain. The resulting conceptual domain model is used as a means for the product fragment management, change impact assessment. A resulting log of confirmed change impacts serves as a means to refine and establish direct dependency links between product fragments, in order to enable product traceability.

Goals	Common reference point (space)	Common conceptualization	Product fragment management	Product traceability
Means	Externalised knowledge as conceptual model	Modelling framework	Conceptual domain model	Log of confirmed change impact

Figure 1.3 Overview of goals and corresponding means

1.6 Way of Working and Results

The research method applied in this research is a design research. Considering the research methodology in that context, the way of working consists of a descriptive analysis phase, a normative development, implementation phase and an empirical evaluation phase. All together the phases include the following six steps.

- The survey of the state-of-the-art step includes an investigation of information systems engineering life-cycle methods, existing methods for product traceability, model management, conceptual modelling and an analysis of the process of semantic enrichment.
- The analysis of the requirements step includes an inventory of problems with regard to distributed collaborative work.
- The development of the approach step includes a specification of components of the approach and stepwise instructions for its use.
- The development of the algorithm step includes definition of algorithms for a semantic similarity calculation and change impact assessment.
- The prototype application step includes development and implementation of a prototypical environment for collaborative modelling and product fragments management based on the results of the previous steps.
- The empirical evaluation step includes experimental evaluation of the approach and proposed algorithms based on observations from the case studies.

The descriptive analysis phase resulted in a framework for product fragment management and a set of requirements for both model fragment and product fragment management in a distributed development. The problem statement and current issues were described in (Strasunskas, 2002). The requirements were documented in (Strasunskas et al., 2003) and the initial framework in (Strasunskas, 2003). The normative development resulted in a novel method for product fragment interrelation, change impact prediction and direct dependency links establishment. The initial design was documented in (Strasunskas et al., 2004). The implementation phase resulted in a prototype implemented in Python, wxPython programming languages and PostgreSQL ORDBMS. Earlier version of prototype repository system implementation is reported and documented in (Fidjestol, 2005). Calculation of similarity between two product fragments were investigated using Bayesian Belief Networks, computing probability of change impact based on the relationships in (Strasunskas & Hakkarainen, 2004), and weighted graphs, calculating the semantic distance between fragments in (Strasunskas & Hakkarainen, 2003). A part of the approach for distributed collaborative modelling has been described in Strasunskas & Lin (2005) and Strasunskas et al. (2006). In addition, two master students contributed to the refinement of ideas presented in this thesis. Erichsen (2003) has analysed traceability focusing on applicability of reference model and direct links for that purpose. Lau (2004) has investigated project content management and browsing using RDF.

1.7 The Structure of the Thesis

The thesis is divided into nine chapters, where this introduction has outlined the objectives, problem definition, approach and contributions. In the next part of the thesis, chapters 2 and 3 provide the background and context for the work. The third part of the thesis, chapters 4 to 6, elucidates the contributions of the thesis, while chapter 7 outlines a realisation of our method and chapter 8 evaluates the method and the implementation. Finally, chapter 9 concludes the thesis and discusses future research directions.

Chapter 2 – Cooperative and distributed information systems engineering provides detailed theoretical background and context for this thesis. This chapter provides an overview of distributed development and main features of cooperative product development, analyses the role and place of conceptual model in development life-cycle. Main requirements underlying our approach are listed here.

Chapter 3 – State-of-the-art surveys is a technological overview of the state-of-the-art tool support in the area of modelling (CASE), computer supported collaborative work (CSCW), repositories including content management systems and supportive techniques.

Chapter 4 – Repository objects presents an overall method first, then the basic concepts and essential techniques supporting the proposed method are discussed. Namely, storage of repository objects, namespace and versioning framework.

Chapter 5 – Model fragment management presents a part of the method for collaborative modelling, elucidates a method for model fragment, model configuration management, provides detailed descriptions of modelling activities achieving common conceptualisation of a domain.

Chapter 6 – Product fragment management specifies a part of the method for product fragment management in more detail, discusses change impact assessment, establishment of direct dependency links based on confirmed and observed change impacts.

Chapter 7 – Realisation of the method outlines an architecture and realisation of the method. The prototype has been implemented to verify whether the earlier described method is applicable solution. Discussion in this chapter is focused in functionality specification rather than technical details.

Chapter 8 – Evaluation of the method discusses different possible evaluation alternatives and methods, argues for the chosen evaluation method. Two cases were used for an experiment, the performance of the prototype and proposed method was compared and evaluated against one commercial tool and a traditional technique. Data from the experiment were gathered from used tools as result of the performed tasks and in a form of a post-evaluation questionnaire.

Chapter 9 – Conclusions concludes the thesis and reflects on the process as well points out future directions.

Appendix A – Referent Model Language presents RML – a concept modelling language used as experimental and illustrational modelling language in this thesis.

Appendix B – Prototype visualisation includes an overview of graphical user interface and functionality implemented. It supplies additional material for chapter 7.

Appendix C – Questionnaire includes the questionnaire used for an experiment described in chapter 8.

Appendix D – Experimental materials presents description of the cases used in the experiment, in a form they were given to the test subjects; lists the product fragments used in both cases and illustrates typical product fragments.

Appendix E – Data collected, provides the raw data collected in the experiment.

Appendix F – Collection of Papers provides the papers written by the author of this thesis and referenced in section 1.6.

2 Cooperative and Distributed Information Systems Engineering

"It is through cooperation, rather than conflict, that your greatest successes will be derived..."
– Ralph Charell

This chapter defines and overviews main features of cooperative IS (product) engineering, discusses collaboration and cooperation activities, provides an analysis of the product role in distributed development; overviews the main ways of exchanging product fragments and explicating their meaning. An analysis of the role of domain and conceptual modelling in the product development methodologies (life-cycle) is in focus here. The analysis is based on a review of the literature on the aspects of IS engineering and it is proven that the conceptual domain model plays an important role as a *medium for cooperation, product and project management* in distributed product development projects.

Finally, an importance of supporting techniques as configuration management, version control is shortly discussed. A set of requirements for supporting collaboration in conceptual domain model centric product development environments is derived from the discussion and presented before summarizing the chapter. But first, some basic concepts used in this chapter and the rest of the thesis are defined in the following section.

2.1 Information System Engineering

The product (software system) and its developers are in focus in this chapter. We use the term *product* when we refer to *the product being developed* by the development project, see Figure 2.1. The product contains all requirements and design documents, test scenarios, models, sketches, minutes, notes, source code, etc. that are created and often updated throughout the development project. The product itself plays a "dual role" in the development process (Seltveit, 1994). The product is used for supporting communication and understanding among the developers in the development project. The same product and its different configurations are later used for (automatically or semi-automatically) manufacturing the end-product in form of executable software system. The product is also regarded as an "anchor" for supporting cooperation among the developers (Farshchian, 2001).

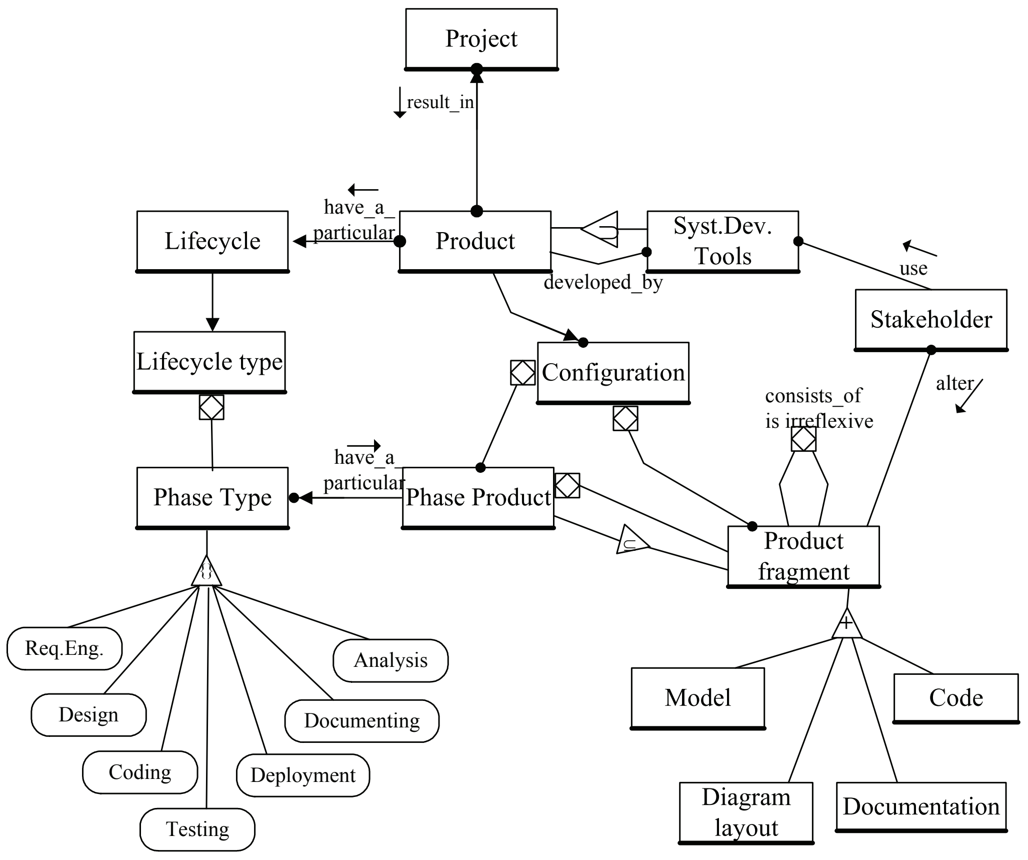


Figure 2.1 Product development[2]

We use the term *stakeholder* to denote all the people who are interested in, affected by the product, or play a significant role in the product development. Stakeholders can be analysts, domain experts, project managers, coordinating authorities, programmers, testers, end-users, etc. A *product development project*, or project for short, is a team that is actually performing the product development. A project consists of stakeholders with different view of the world, because of different educational, cultural background, various experiences, as classified by Agerfalk et al. (2005) into dimensions of distribution: geographical distance; temporal distance; socio-cultural distance. A *system development tool*,

[2] In Referent Model Language (RML) notation, for detail description see Appendix A.

or development environment, is a technical infrastructure, tool, or technology that is used to support the developers in a project.

The product is seen as constituted by a set of interconnected artefacts called *phase product*. Examples of phase products are requirements specifications used for documenting stakeholders' requirements, user interface for demonstrating, test-scenarios, formal and semi-formal models describing the problem domain, design documents describing the technical design of the computer system, source code files written in a programming language, etc. Phase products do not exist in isolation, but their meaning is normally defined in relation to other product objects. There exist different relations among *product fragments*. Examples of relations are dependency relations (e.g., composed_of, based_on, derived_from), import relations, "part_of" relations, etc. A product is a specific *configuration* of product fragments. A configuration will normally include a subset of all the available product objects. *Product development* is the process of creating and updating these product fragments and relations (configurations), requiring an intensive cooperation among the stakeholders.

2.2 Team and Product in Distributed Product Development

In this thesis we endeavour to support work of a geographically dispersed team, therefore, in this section we discuss a coordination aspect of teamwork and investigate whether ISD is a cooperative or collaborative activity. Next, we discuss the role of the product itself in the development process and the way it supports teamwork.

2.2.1 Coordination of teamwork

There are many authors whom simply consider both terms as synonyms and use them interchangeably, e.g., in WordNet (2005) cooperation is a kind of collaboration, and vice versa, i.e., both terms are hyponyms of each other. Some authors do distinguish them as two different terms, but without drawing a clear line in between. For instance, definitions provided by a collaborative community in Wikipedia[3] (2005), where "cooperation refers to the practice of people or greater entities working in common with commonly agreed-upon goals and possibly methods, instead of working separately in competition" (Wikipedia/Cooperation, 2005). While "collaboration is simply defined as working together with one or more others" (Wikipedia/Collaboration, 2005). Eventually, others (e.g., Dillenbourg et al. (1996)) draw an explicit distinction between them:

> *"Cooperation and collaboration do not differ in terms of whether or not the task is distributed, but by virtue of the way in which it is divided: in cooperation, the task is split (hierarchically) into*

[3] *Wikipedia* is a Web-based, multi-language, free-content encyclopedia written collaboratively by volunteers and sponsored by the non-profit Wikimedia Foundation.

independent subtasks; in collaboration, cognitive processes may be (heterarchically) divided into intertwined layers. In cooperation, coordination in only required when assembling partial results, while collaboration is "... a coordinated, synchronous activity that is the result of a continued attempt to construct and maintain a shared conception of a problem" (Roschelle & Teasley, 1995)."

Thereinafter, we treat ISE in this thesis as a cooperative activity, where different parties work on different parts of a product in question. However, we distinguish the problem and domain analysis phase of systems development, where all stakeholders need to participate and construct together a shared conceptualization of a problem. Therefore, conceptual domain modelling is treated as a collaborative activity.

2.2.2 Role of product in distributed development

The product being developed is a central part of any product development project (Farshchian, 2001). As the success of the whole project will usually be assessed by the quality of the product. Most product development environments, such as CASE (Computer Aided Software Engineering) and ISEE (Integrated Software Engineering Environment) tools, are built around a product (which is often stored in a central repository). Interacting with the product takes a considerable part of the activities of the developers.

A project consists of stakeholders who usually view the world in their own specific way. This specific view is referred to as the local reality. The local reality is the way the world is for the particular stakeholder. In the project stakeholders externalize their local reality, by communicating their view and understanding. In this way they participate in a construction of an organizational (common) reality. Then stakeholders reflect on constructed organizational reality and adjust their own local reality. This process is iterative and simultaneous (see Figure 2.2).

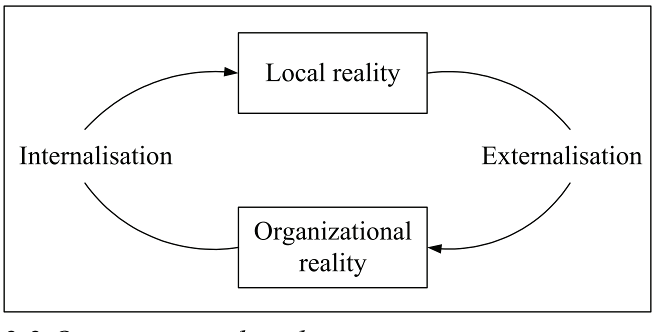


Figure 2.2 Organizational reality construction in an organization (Gjersvik, 1993)

Product knowledge does not exist in the beginning of a project. Therefore, in geographically distributed projects it is very important (taking into account different background, culture, etc. of the participants) to develop common understanding of the problem in question. Thus, we see it important for the involved stakeholders to externalize their *knowledge of problem domain* first. Every stakeholder should have a means to express and communicate own understanding of the problem on-hands. Means should be provided to perform explicit conceptualisation to efficiently and effectively negotiate and specify concepts they use (Proper & Hoppenbrouwers, 2004).

Farshchian (2001) distinguishes three essential properties of a product developed in distributed environment, namely, *externalized knowledge*, *boundary object*, and *coordination mechanism*. They are summarized in Table 2.1.

Table 2.1 Three properties of product important for supporting cooperation (Farshchian, 2001)

Property	**Support for cooperation**	**The role of physical proximity**
Product as externalized knowledge	Supports cooperative learning, criticism, creativity.	Shared physical space: - embodies the developers and the product; - supports continuous exchange of information related to the product; - supports flexible and customized interaction between the developers and the product.
Product as boundary object	- Supports understanding across different communities of practice; - facilitates negotiation of local understandings; - supports information sharing.	Shared physical space: - allows the developers to customize the product to theirs local needs; - offers low-cost and dynamic communication channels for resolving misunderstandings.
Product as coordination mechanism	Supports coordination of daily activities of developers.	Shared physical space: - allows continuous access to information about modifications to the product; - supports access to information about the process through which these modifications are made.

First, a product is an externalized knowledge. Stakeholders cooperate in order to externalize their knowledge, later resulting in the product. Second, a product is a boundary object. A development project usually consists of stakeholders with varying domain knowledge, background and experience. The same product is often used by all these people in order to support the different local understandings of accumulated knowledge. Third, a product is a coordination mechanism, where information about the status of the product is used by the stakeholders in order to coordinate their work (Farshchian, 2001).

2.2.3 Exchange of product fragments

Multiple tools usually are required to specify all aspects of a system. Exchanging data among tools is difficult due to the absence of accepted tool independent data formats. There are plentiful of data interchange formats such as CDIF (Gray & Ryan, 1997), XIF (Microsoft/XIF, 1999), XMI (OMG/XMI, 1998), SPOOL (St-Denis et al., 2000), UXF (Suzuki & Yamamoto, 1998). Most of them are based on eXtensible Mark-up Language (XML). Furthermore, in collaborative projects it is often the case that partner organizations use different tools for accomplishing the same task. Automated information exchange is also in this case obstructed by the lack of tool support for data exchange. Information sent for further refinement or analysis in tools within or outside the originating organization often have to be re-entered manually into the receiving organization's tool-set, – a tedious and error-prone process.

The cost-efficient sharing of data between heterogeneous tools and repositories requires the adoption of a standard for an industry-wide data interchange format. Product fragments are stored and interchanged through a repository.

Currently undergoing research in semantic Web (Berners-Lee et al., 2001) area will contribute with a new data (knowledge) interchange format. As it is now, the most of data interchange formats are based on XML or XML dialect. In this thesis we are not trying to contribute with any improvement or suggestion regarding data interchange format. We relay on the tools ability to produce XML file of a development object, and then only adaptable XML parser matters.

Communicating the meaning

Sharing of information among project members is normally done by collecting and organizing the information needed with respect to the task on-hand. In the Computer Supported Cooperative Work (CSCW) literature, this is usually referred to as a "common information space":

> *"Here, focus is on how people in a distributed setting can work cooperatively in a common information space – i.e. by maintaining a central archive of organizational information with some level of 'shared' agreement as to the meaning of this information (locally constructed), despite the marked differences concerning the origins and context of these information items." (Schmidt & Bannon, 1992)*

Based on the above cited definition, system should provide storage facilities for project information (product fragments), and project members need to have means for reaching "some level of 'shared' agreement as to the meaning of this information". Usually different members will have their own interpretation of the meaning of information. This information must be explicit and communicated to other members of the project. Individual or domain interpretations are negotiated and related to each other until the desired level of shared agreement is reached (Bannon & Bodker, 1997), i.e., "harmony" between local and organisational realities is achieved.

Tasks in the geographically distributed projects are usually assigned based on competence of the involved parties, i.e., taking an advantage from distributing product development based on skills of the participants. Each stakeholder involved in ISE develops own product (fragment) using his/ her preferred representation. At a certain time, the products developed in parallel must be integrated; discrepancies and similarities must be detected through the communication and conversation among the people involved. Changes to the products have to be made according to unresolved discrepancies. How do they communicate product fragments meaning with colleagues, if some of them are not familiar with a specific notation, the product fragment is represented in?

Since direct communication is not a trivial way to do in geographically distributed projects (because of time difference, language barriers, etc.), the stakeholders need to find a way to enhance semantics of the product fragments.

Metadata

One of "traditional" ways to clarify semantics is specifying additional data about data, i.e., metadata. Metadata is information on the organization of the data, the various data domains, and the relationships between them (Baeza-Yates & Ribeiro-Neto, 1999). Metadata allows systems to collocate related information, and helps users find relevant information. Usually metadata is differentiated between descriptive metadata, i.e., metadata which is external to data meaning, and pertains how the data was created; and semantic metadata, i.e., characterizes the subject matter of the data content.

Traditionally, metadata is created by professionals or authors. Professionally created metadata are often considered being high quality, but costly to produce. While author created metadata is more scalable but still has a problem as being disconnected from intended and unintended users (Mathes, 2004).

Content creation applications (word processors, Webpage creation tools, etc) often have facilities for author-supplied attributes or automated capturing of attributes that simplify the creation of metadata. As these facilities grow more sophisticated, it will be easier and more natural to combine application-supplied metadata (e.g., creation dates, tagged structural elements, file formats) with creator-supplied metadata (e.g., keywords, authors, affiliations). Combination of those attributes increases the quality and reduces the cost of metadata descriptions.

Meanwhile, specifying metadata is perceived as additional burden for developers, and is not extensively used. The only available metadata is application-supplied metadata: creation, modification dates, file formats, user name, etc. which reflects only descriptive metadata and do not facilitate communication of the meaning contented in the development object.

Furthermore, there are plentiful of metadata standards and formats for various reasons and a certain usage domain. Below we survey some of them.

Dublin Core is the most common standardisation initiative proposal (Weibel et. al, 1998) for a "core set of elements" (Borgman, 2000) proposing 15 basic elements of a description. The fifteen elements of the Dublin Core Metadata Element Set (Title, Creator, Subject, Description, Publisher, Contributor, Date, Type, Format, Identifier, Source, Language, Relation, Coverage, and Rights) are the defining feature of Dublin Core.

CWM (Common Warehouse Metamodel) is a specification that describes metadata interchange among data warehouses, knowledge management and portal technologies. It provides a framework for "representing metadata about data sources, data targets, transformations and analysis, and the processes and operations that create and manage warehouse data and provide lineage information about its use" (CWM, 2005).

UDDI (Universal Description, Discovery, and Integration) is a standard for locating web services by enabling robust queries against rich metadata. Metadata about web-services are stored in repositories. The information provided in a listing consists of three conceptual components: "white pages" of company contact information; "yellow pages" that categorize businesses by standard taxonomies; and "green pages" that document the technical information about services that are exposed (UDDI, 2005).

ebXML (Electronic Business using eXtensible Markup Language), is a modular suite of specifications that is designated to enable enterprises despite of a geographical location to conduct business over the Internet. The ebXML specification provides a standard infrastructure for sending business messages across the internet (ebXML, 2005).

RDF (Resource Description Framework) is a specification developed by the World Wide Web Consortium (W3C). RDF defines a uniform mechanism for describing resources, and makes no assumptions about a particular application domain. It is based on simple data model for representing named properties and property values (Miller et al., 2005):

• *Resources.* All things that are described by RDF expressions are called resources. A resource is anything that can be identified by the use of an URI. Examples are a web page, a person.

• *Properties.* A property is a specific aspect or characteristic used to describe a resource. The definition of a property is not a part of the core RDF model but can be defined by the use of the RDF Vocabulary Description Language (RDF Schema), which is a specification for the formal declaration of the resource classes and properties.

• *Statements*. A specific resource together with a named property plus the value of that property is an RDF statement. These three individual parts of a statement are called the subject, the predicate, and the object respectively. The object of a statement (the property value) can be another resource identified by a URI or it can be a literal like a simple string or number.

The structure of any expression in RDF can be viewed as a directed graph that consists of nodes and labelled, directed arcs that link pairs of nodes. The statements of RDF are directed, and a property only captures one side of the relationship semantics – as it is seen from the subject.

RDF statements are the fundamental mechanism for expressing metadata (Miller et al., 2005). Statements provide a very generic and flexible way of expressing metadata. Initially there is no RDF model. However, it is possible to express statements without a schema. Schema-conformance only becomes important in application-specific settings, for instance, for query languages that exploit schema information in order to provide structural queries. A statement is simply a triple – subject – object –predicate – that assigns a property to a resource, as follows.

- The subject is always an rdfs:Resource. The resource is identified by the URI pointing to it, or rather to the metadata document describing it.
- The predicate denotes the property being assigned to the resource. A property in RDF is both considered an attribute of the resource and possibly a relation between two resources. The rdf:Property class is a subclass of the rdfs:Resource class.
- The object of a statement is the value of the property. Objects are either other resources or literal values. In the latter case, the statement is often called a lexical statement.

Ontology

A different way of explicating the meaning of the content is by use of ontology, though could be seen as metadata specification as well. Recently, ontologies have been advocated as a means for gathering and formalising application domain knowledge in order to make it available for human analysts as well as for automated knowledge processors. According to Gruber's definition (1993), an ontology is "an explicit specification of a shared conceptualisation".

According to Guarino (1998) there are different kinds of ontologies: top-level ontologies, that describe very general concepts like space, time, matter, event, etc.; domain ontologies, that describe the vocabulary related to a generic domain (like medicine, or automobiles); task ontologies, that describe generic tasks or activities (like diagnosis or selling); and finally, application ontologies, that describe concepts depending on a particular domain and task. Application ontologies are specializations of both the domain and task ontologies.

Main benefit of ontologies is an identification of specific classes of objects and relations that exist in some domain. The main purpose of developing ontologies is to clarify the domain's structure of knowledge and to enable knowledge sharing and reuse (Chandrasekaran et al., 1999). In addition ontologies in distributed development would facilitate:

- Consensus knowledge of a community of people;
- High expressiveness, enabling the ontology users to say what they wish to say;
- Coherence and interoperability of resulting knowledge bases;
- Stability and scalability of ontologies.

In the simplest case an ontology describes a hierarchy of concepts related by subsumption relationships, while in more sophisticated cases, suitable axioms are added in order to express other relationships between concepts and to constrain their intended interpretation (see Figure 2.3). Ontologies are consensual and formal specifications of a vocabulary used to describe a specific domain (Decker et al., 1999). Ontologies are usually used as an "explicit specification of a conceptualization" (Gruber, 1993). Therefore, an application of ontologies is in an integration task to describe the semantics of the information resources, i.e. explicitly describe content. With respect to the integration of data sources, ontologies can be used for the identification and association of semantically corresponding information concepts (Wache et al., 2001). Other application areas use the ontology as the global query schema.

Obviously, ontologies have been applied for diverse purposes. Gruninger and Lee (2002) summarise main usage of ontologies:

- For communication:
  - between implemented computational systems;
  - between humans;
  - between humans and implemented computational systems.
- For computational inference:
  - for internally representing and manipulating plans and planning information;
  - for analyzing the internal structures, algorithms, inputs and outputs of implemented systems in theoretical and conceptual terms;
- For reuse (and organization) of knowledge:
  - for structuring or organizing libraries or repositories of plans and planning and domain information.

Ontologies are said to be useful for developing methods and tools in the context of requirements elicitation and engineering for information systems, natural language processing and, especially, semantic Web services by contributing to the quality of interoperating systems, and by reducing costs (Mayr, 2002).

Use of ontologies to provide semantic interoperability in information sharing has been long realized (Gruber, 1991; Kashyap & Sheth, 1994; Wache et al., 2001). By mapping concepts, terms and various information resources to ontological concepts, it is possible to explicitly define the semantics of that resource in particular domain.

A lot of work is done on the study of formal ontology in general (Wiederhold, 1994; Guarino & Poli, 1995) as well as their application to ensure interoperability in heterogeneous information systems (Wiederhold, 1994; Kashyap & Sheth, 1994; Mena et al., 1996).

2.2.4 Conceptual model vs. ontology

Ontology as philosophical discipline deals with studying the nature of being, reality, and substance. Here we justify our view "conceptual model = ontology", in a sense of their applications in computer science. That is done in order to clarify terminology used and justify the state-of-the-art review in next chapter.

Definition of ontology is really broad. A good illustration of whole spectrum of ontology is Figure 2.3, adopted by Krogstie et al. (2006) from (Daconta et al., 2003). Krogstie et al. (2006) points out that calling all of the terms on the left hand side an "ontology" brings confusion about the word "ontology", though some authors does. The difference here between knowledge models is in power of semantics expressiveness.

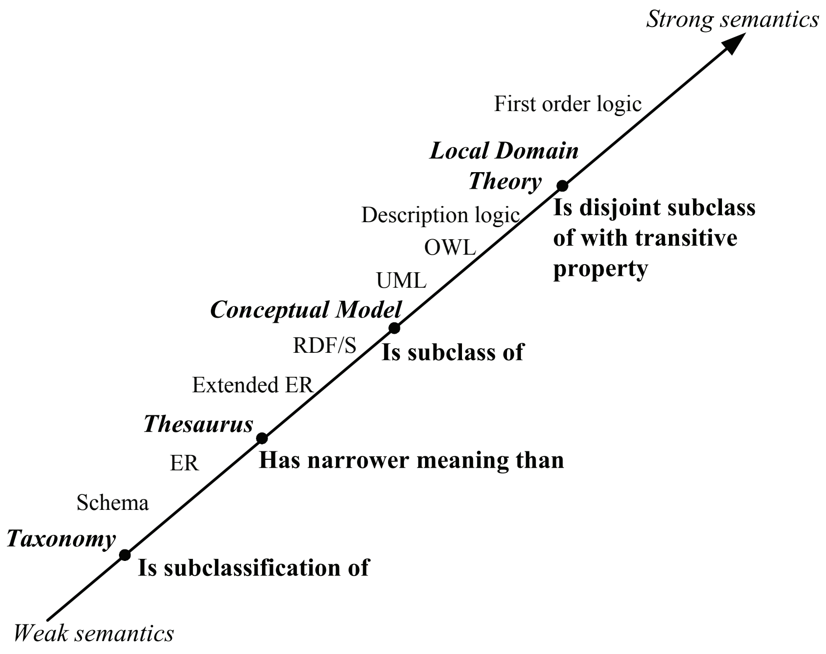


Figure 2.3 Ontology spectrum (adopted from Krogstie et al. (2006))

Ontologies, or explicit representations of domain concepts, provide the basic structure of the domain. Ontology defines the vocabulary of a problem domain and a set of constraints on how terms can be combined to model the domain. In a distributed environment, agents (human and computer) use ontologies to establish communication at the knowledge level using specific languages and protocols. Fensel (2001) defines ontology as "*a shared and common understanding of a domain that can be communicated between people and application systems*". Ontologies are explicit representations of agents' commitments to a model of the relevant UoD; hence they enable knowledge sharing and reuse (Devedzic, 2002).

Jasper and Uschold (1999) identify four main categories of ontology application scenarios, as follows.

Neutral Authoring: An information artefact is authored in a single language, and is converted into a different form for use in multiple target systems.

Ontology as Specification: An ontology of a given domain is created and used as a basis for specification and development of some software.

Common Access to Information: Information is required by one or more persons or computer applications, but is expressed using unfamiliar vocabulary, or in an inaccessible format. The ontology helps render the information intelligible by providing a shared understanding of the terms, or by mapping between sets of terms.

Ontology-Based Search: An ontology is used for searching an information repository for desired resources (e.g., documents, web pages, names of experts).

A similar purpose convey a conceptual model in IS engineering. A conceptual model is used both for communication and representation. More specifically the use of conceptual model in IS development is identified and summarized in (Krogstie & Solvberg, 2000) as follows.

Representation of systems and requirements: The conceptual model represents properties of the problem area in addition to perceived requirements for the information system. The conceptual model can give insight into the problems motivating the development project, and can help the systems developers and users understand the problem domain.

Vehicle for communication: The conceptual model can serve as a means for sense-making and communication among stakeholders. It facilitates a more reliable and constructive exchange of opinions between users and the developers of the IS, as well as between different users.

Basis for design and implementation: The conceptual model can act as a prescriptive model, to be approved by the stakeholders who specify the desired properties of a system in question. The model can establish the content and boundary of the UoD.

Documentation and sense-making: The conceptual model is a documentation of the systems that are in use in the organization. Due to its independence of the implementation, it is less detailed than other representations, while still representing the basic functionality of the system. Compared to manually produced textual documentation, the conceptual model is easier to maintain since it is constructed as part of the process of developing and maintaining the application system.

Consequently, we will treat the conceptual model and ontology in this thesis as equal, but use a term conceptual model, with exception where cited authors use term ontology explicitly. Though, we are aware of different philosophical meaning and purpose of ontology, where philosophical ontology seeks a classification that is exhaustive in the sense that all types of entities are included in the classification.

2.3 Role of Conceptual Modelling in IS Engineering

"Conceptual modelling is the first step and one of the most important steps for application engineering" Chen et al. (1997, p. 297)

Here we survey the main existing IS engineering methodologies in order to investigate when in a life-cycle of ISE a conceptual domain model is constructed and how it is used.

An important stage in the construction of Information Systems is a modelling. Where the relevant, meaningful information structures in a domain are determined and documented in an accurate and unambiguous way. Traditionally information system engineering has made the assumption that IS captures some excerpt of world history and hence has concentrated on modelling information about the Universe of Discourse (Olle et al., 1988).

The traditional way of engineering information systems is through conceptual modelling which produces a specification of the system to be developed (Rolland & Prakash, 2000), see Figure 2.4. Such specification acts as a prescription for system construction. A conceptual domain model is an abstract representation of the real world phenomena that are of relevant to a project. The conceptual model is usually constructed during the system analysis phase (Solvberg & Kung, 1993).

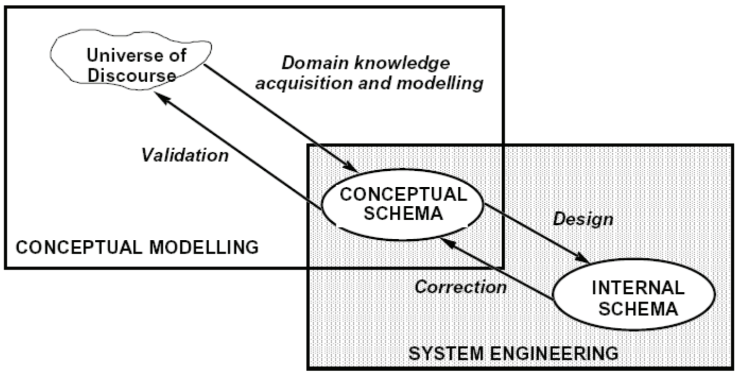


Figure 2.4 Simplified two-phase organisation of system life-cycle (Rolland & Prakash, 2000)

The use of conceptual model (ontology) to provide underpinning for information sharing, heterogeneous database integration, and semantic interoperability has been long realized (Gruber, 1991), (Kashyap & Sheth, 1994), (Wache et al., 2001).

Olive (2005) defines a "conceptual schema-centric development" (CSCD) as central issue in order to revive the goal of automated information systems building. Olive argues that conceptual schema is necessary to develop an information system, and, therefore, the CSCD approach does not place any extra burden on developers. Olive (2005) defines the Principle of Necessity: "To develop an information system it is necessary to define its conceptual schema".

2.3.1 Perspectives of IS Engineering

Though, conceptual modelling ("mind alignment") is an important step in ISE, there is no common view how this should be done and this results in a various perspectives towards IS engineering. A development method (development life-cycle) is a set of rules, approaches and tools to support development of a product. (Krogstie, 1995) describes a methodology classification framework consisting of seven categories, namely *weltanschauung*, *coverage in process*, *coverage in product*, *reuse of product and process*, *stakeholder participation*, *representation of product and process*, and *maturity*.

Weltanschauung describes the underlying philosophy or view to the world. It is examined why the product construction is addressed in a particular way in a specific methodology. In the FRISCO report (Falkenberg et al., 1997), three different views are described, namely the objectvistic, the constructivistic and the mentalistic view. *Objectivistic view* claims that reality exists independently of any observer. The relation between reality and the model is trivial or obvious. *Constructivistic view* claims that reality exists independently of any observer, but what each person possesses is a restricted mental model only. The relationship between reality and models of this reality are subject to negotiations among the community of observers and may be adapted from time to time. *Mentalistic view* claims that reality and the relationship to any model is totally dependent on the observer. We can only form mental constructions of our perceptions. In many cases, when categorizing a method, the Weltanschauung will not be stated directly, but exist indirectly.

Coverage in process concerns the method's ability to address *planning for changes*, *single and co-operative development* of product, which includes analysis, requirements specification, design, implementation and testing, *use and operations* of product, *maintaining and evolution* of product, and *management of planning, development, operations and maintenance* of products.

Coverage in product is described as the method concerns planning, development, usage and maintenance of and operates on *one single product*, *a family of related product*, *a whole portfolio of products* in an organization, and *a totality of the goals, business process, people and technology* used within the organization.

Reuse of product and process is important to avoid re-learning and recreation. A method may support reuse of product or reuse of method as processes. There are the following six dimensions of reuse. *Reuse by motivation* answers the question - why is reuse done? Different rationale may be for example productivity, timeliness, flexibility, quality, and risk management goals. *Reuse by substance* answers the question – what is the essence of the items to be reused? A product is a reuse of all the deliverables that are produced during a project, such as models, documentation and test cases. Reusing a development or maintenance method is process reuse. *Reuse by development scope* answers the question – what is the coverage of the form and extent of reuse? The scope may be either external or internal to a project or organization. *Reuse by management mode* answers the questions - how is reuse conducted? The reuse may be planned in advance with existing guidelines and procedures defined, or it can be ad-hoc. *Reuse by technique* answers the question - how is reuse implemented? The reuse may be compositional or generative. *Reuse by intentions* answers the question - what is the purpose of reused elements? There are different intentions of reuse. The elements may be used as they are, slightly modified, used as a template or just used as an idea.

Stakeholder participation reflects the interests of different actors in the ontology building activity. The stakeholders may be categorized into those *responsible for developing* the method, those with *financial interest* and those who have *interest in its use*. Further, there are different forms of participation. *Direct* participation means every stakeholder has an opportunity to participate. *Indirect* participation uses representatives, thus every stakeholder is represented through other representatives that are supposed to look after their interests.

Representation of product and process can be based on linguistic and non-linguistic data such audio and video. Representation languages can be *informal*, *semi-formal* or *formal*, having a logical or executional semantics.

Maturity is characterized on different levels of completion. Some methodologies have been used for a long time; others are only described in theory and never tried out in practice. Several conditions influence maturity of a method, namely if the method is *fully described*, if the method lends itself for *adaptation, navigation and development*, if the method is used and updated through *practical applications*, if it is *used by many organizations*, and if the method is *altered* based on experience and scientific study of its use.

There are even more IS engineering methods. Next section discusses the main development methodologies in the light of conceptual model usage.

2.3.2 System development methodologies

ISE methods usually "differ greatly, often addressing different objectives" (Avison & Fitzgerald, 2002). ISE is perceived as not only technical development, but includes social aspects of development, which usually are not supported by traditional methods (Kiely & Fitzgerald, 2003). Methods are typically developed to make Information System engineering process more controllable.

There are basic phases that constitutes IS engineering. In fact, the major phases of a system development project are usually system analysis, system design and system implementation (Solvberg & Kung, 1993). Generic software development activities are also identified by Loucopoulos and Karakostas (1995), Sommerville (1992). These activities include software specification, software development, software validation, and software evolution. Various compositions, decompositions and iterations of these activities (major phases) and sub-activities focusing on different needs resulted in many development life-cycles.

More specifically, Solvberg and Kung (1993) describes the lifecycle of a project as comprising the following 8 phases: Pre-project study; Requirements specification; System modelling and evaluation; Functional specification; Data processing system architecture; Programming; System installation; Project evaluation.

The ISO 12207 Software Engineering Standard (SEPT, 2005) describes a meta-model for software engineering life-cycle processes that consists of thirteen activities that can be mapped onto a chosen life-cycle model. The first activity, "process implementation", is related to starting the methodology itself, while another four of the activities are system related: *system requirements analysis*; *system architectural design*; *system integration* and *system qualification*. The remaining eight are related to the software itself and the standard notes that "these activities and tasks may overlap or interact and may be performed iteratively or recursively". Short descriptions of these eight remaining activities obtained from the IEEE Standard Glossary of Software Engineering Terminology (IEEE, 1990) are:

– **requirements analysis**, the process of studying user needs to arrive at a definition of system, hardware, or software requirements.

– **architectural design**, the process of defining a collection of hardware and software components and their interfaces to establish the framework for the development of a computer system.

– **detailed design**, the process of refining and expanding the preliminary design of a system or component to the extent that the design is sufficiently complete to be implemented.

– **coding and testing**, where coding is defined as "… the process of expressing a computer program in a programming language" and testing is "the process of analyzing a software item to detect the differences between existing and required conditions (e.g., bugs) and to evaluate the features of the software items".

– **integration**, the process of combining software components, hardware components, or both into an overall system.

– **qualification testing**, testing conducted to determine whether a system or component is suitable for operational use.

– **installation**, the period of time in the software cycle during which a software product is integrated into its operational environment and tested in this environment to ensure that it performs as required.

– **acceptance support**, formal testing conducted to determine whether or not a system satisfies its acceptance criteria and to enable the customer to determine whether or not to accept the system.

Though, classical life-cycle method is the "waterfall" model proposed by Royce (1987), which consists of seven steps or phases that proceed in a linear way: *System Requirements*; *Software Requirements*; *Analysis*; *Program Design*; *Coding*; *Testing*; and *Operations*. The waterfall model focuses heavily on the documentation produced during each implementation phase and there may be some iteration between successive steps.

The "V" model is a variant of the waterfall model where each step down the left hand side of the "V" has a corresponding validation or verification step on the right hand side (see Figure 2.5). This model presents the opportunity for more "formal" development where documents from the left hand-side feed into the validation activities on the right. Where the left tail of the "V" represents the specification stream where the system specifications are defined. The right tail of the "V" represents the testing stream where the systems is being tested (against the specifications defined on the left-tail). The bottom of the "V" where the tails meet, represents the development stream. While the spiral model (Boehm, 1987) is another alternative life-cycle that includes: risk analysis, planning, engineering and customer evaluation. Starting in the centre of a spiral the developers work through a planning phase, followed by risk analysis, the engineering of a prototype system and then customer evaluation. The cycle then repeats and each move around the spiral progresses outwards towards the final system in an evolutionary way.

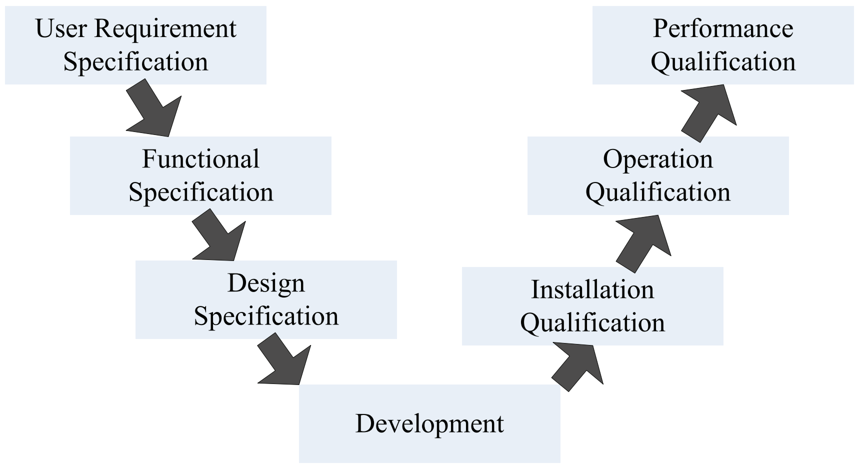

User Requirement Specification
Functional Specification
Design Specification
Development
Installation Qualification
Operation Qualification
Performance Qualification

Figure 2.5 "V" lifecycle model

Twin Peaks model (Nuseibeh, 2001) is a concurrent, spiral development process suggests a partial development model that highlights the concurrent, iterative process of producing progressively more detailed requirements and design specifications. This model emphasizes the equal status of the specification of requirements and architectures (Figure 2.6).

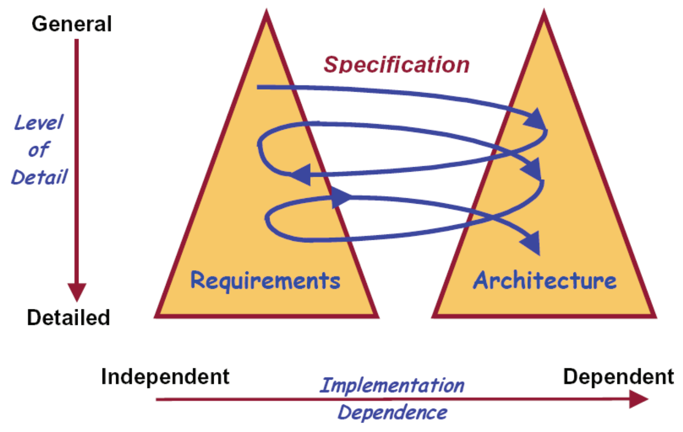


Figure 2.6 Twin Peaks – a model of concurrent development

Model allows early exploration of the solution space, thereby allowing incremental development and the consequent management of risk; rapid and incremental requirements identification and architectural matching; and focuses on finer-grain development and is therefore more receptive to changes as they occur.

Widely adopted Rational Unified Process (RUP) is a software development process, which, as claimed, comprises all parts of a complete software development process, both from a technical and managerial point of view. RUP defines a technical and managerial lifecycle of a software system as iterations of the phases. The managerial phases are: inception, elaboration, construction, transition and evolution, whereas the software development process itself is defined as; planning, analysis, architecture, design, implementation, integration and testing (IBM, 2005b). The phases may be iterated as needed in order to satisfy the requirements. RUP also defines the artefacts which are to be produced in each phase, and what they should include.

Zachman (1987) defines a framework that lays down the main views that are necessary to specify during ISE. Figure 2.7 illustrating Zachman's framework should be read from top-left to bottom-right corner.

	Structure (What)	**Activities (How)**	**Locations (Where)**	**People (Who)**	**Time (When)**	**Motivation (Why)**
Objectives/ Scope (Planner's view)	Most significatn business concepts	Mission	International view of where organization operates	Human resource philosophies and strategies	Annual planning	Enterprise vision
Enterprise Model (Business Owner's view)	Business language used	Strategies and hig-level business porcesses	Offices and relationships between them	Positions and relationships between positions	Business events	Goals, objectives, business policies
Model of Fundamental Concepts (Architect's view)	Specific entities and relationships between them	Business functions and tactics	Roles played in each location and relationships between roles	Actual and potential interactions between people	System events	Detail business rules
Technology Model (Designer's view))	System representation of entities and relationships	Program functions/ operations	Hardware, network, middleware	User interface design	System triggers	Business rule design
Detail Representation (Builder's view)	Implementation strategy for entities and relationships	Implementation design of functions/ operations	Protocols, hardware components, deployed software items	Implementation of user interface	Implementation of system triggers	Implementation of business rules
Functioning System	Classes, components, tables, …	Deployed functions/ operations	Deployed hardware, middleware, and software	Deployed user interface (including documentation)	Deployed systems	Deployed software

Figure 2.7 Zachman framework

Model Driven Architecture (MDA, 2003) is one of the recent OMG's initiatives that proposes an approach to system development, i.e. the use of formal models at different abstraction levels, such as, Computation Independent Model (CIM), Platform Independent Model (PIM) and Platform Specific Model (PSM). Transition between abstraction levels is based on transformations between models.

The demand for continuous preservation of software and their quality during long periods leads to a view on software as long-living infrastructure. Evolution of such infrastructures consists of gradual modification steps over all levels of the software development process. In general, every modification step in analysis, design or implementation will require further consistency preserving modification steps within each level of the development process.

In addition to these examples, a number of other life-cycle models exist and the most appropriate model to use for a given project may depend on a number of factors including the type of project, the development style and the organisational maturity of both the developers and the customers. An alternative to the classic life-cycle approaches is to use a meta-model that defines common software engineering activities independently of a particular life-cycle model. Developers can then choose the most appropriate life-cycle for their project and the activities can be mapped onto the chosen model.

Recent focus on agile development brings more methods, which usually are less document-centric and more code-oriented. Therefore, problem modelling is not as important as in other methods, maybe with exception of Agile Modeling (Amber, 2002). Basic of Agile Modeling is to provide a guideline of how to build models and help to resolve possible design problems, but still keeping models simple, i.e., not over-building them.

An extensive classification of different system development methodologies based on earlier discussed classification framework is done by Krogstie (1995). Here we summarize methodologies in respect of whether conceptual modelling is a part of ISE life-cycle and stating when (which phase of life-cycle) it is used (see Table 2.2).

Table 2.2 Conceptual modelling in development life-cycle (extended from Krogstie, (1995))

Methodology	**Role of conceptual modelling**
The waterfall model	Conceptual modelling is applied shallowly if at all.
The structured life-cycle	Conceptual modelling languages are used.
Prototyping	Conceptual modelling is not mandatory, but can be used as a starting point for functional prototyping if the conceptual modelling languages used have a defined operational semantics.
Transformational and operational development	Formal conceptual modelling languages are usually the cornerstone.
METHOD/1	Conceptual modelling is not mandatory, but use of semi-formal conceptual modelling languages is supported in early systems design.
The spiral model	Not specifically supported, but the framework is open for its use.
The hierarchical spiral model	It is based in a large degree on active use of conceptual models, even if no set of concrete modelling languages are mentioned in descriptions.
The fountain model	It is based on object-oriented conceptual modelling.

Methodology	**Role of conceptual modelling**
REBOOT	Conceptual model can be used, especially object-oriented modelling, but the project primarily focused on reuse of detailed design and code.
Multiview	Conceptual modelling is used actively, mainly semi-formal and informal languages.
STEPS	Not mentioned explicitly, object-oriented conceptual modelling is possible.
Zachman framework	Begins with defining main business concepts, then different models describing different enterprise areas are used.
MDA	Model centric. Start from conceptual model (computation independent model) which by the use of transformations is refined to platform specific model.
RUP	Conceptual model is used.
"V"	It is not explicitly stated whether conceptual model is used.
Twin Peaks	Conceptual model is used and aligned with architecture of a system in question.
Agile Modeling	Lightweight conceptual model is used. Model usage here is different then in conventional development. Models here are used to communicate understanding of small part of the system being developed, and models has no later value, i.e., they are "thrown away".

There are many methods for IS development, and different companies are using different life-cycle. However, recent surveys report that many companies claim that they either do not use any methods, or they use own "in-house" developed / adapted methods (Huisman & Iivari, 2002; Kiely & Fitzgerald, 2003). Either way, understanding and documenting the scope of endeavour is seen as an important task for the most companies.

> *"The purpose of domain engineering is to identify, model, construct, catalog, and disseminate a set of software artifacts that can be applied to existing and future software in a particular application domain. As such, it can support the effective and efficient management and development of software assets." (Reinhartz-Berger et al., 2005)*

Benefits from using conceptual domain models are documented in (Hoppenbrouwers et al., 2005a). It is noted that for involved stakeholders domain model provides a terminology and an overview of the scope of the problem to be solved, for developers it provides guidance to make right design decisions. While project managers benefit in planning and controlling the project.

2.4 Supporting Techniques for Cooperative Development

Concurrent engineering changes old practice, when all the required objects were locked during the whole change/ modification activity. Each developer should have direct access to all needed objects. But changed version should be kept with access forbidden for other developers during modification, because the state of fragment is inconsistent in a modification phase. If *n* developers change the same object concurrently, this object should have *n+1* different copies (Estublier, 2001). It means that each developer needs the private copies of fragments. On the other hand, the colleagues know that other changes possibly are done on the same fragments/ objects and want to be incorporated when relevant. In summary, collaborative distributed development needs tools that allow the creation and access to a central composite product, and at the same time support development in local workspaces.

During the life-cycle of ISE, there will be thousands of development objects, with hundreds persons at different sites maintaining and changing them. Development process becomes a continuous history of changes and improvements. To keep all these multi-version, multi-people activities under control, configuration management is needed.

This section discusses issues related with management of product development and modelling. Here we elaborate on change management, modelling aspect and versioning, as well view reconciliation.

2.4.1 Change Management

Configuration management (CM) is the discipline for organizing and controlling evolving systems. Configuration management is an old discipline, born out of systems manufacturing. CM mandates procedures for identification of components and their assemblies, for controlling releases and changes, for recording the product status, and for validating the completeness and consistency of a product (IEEE, 1988). Later CM definitions (Dart, 1991) also include areas like construction management, process management, and team work control. Carnegie Mellon University's Software Engineering Institute (SEI, 1994) defines the purpose of *software configuration (change) management* as establishing and maintaining the integrity of the products of the software project throughout the project's software life cycle.

A standard definition taken from IEEE standard 729-1983 (IEEE, 1987) highlights the following operational aspects of CM:

Identification. An identification scheme reflects the structure of the product, identifies components and their types, making them unique and accessible in some form.

Control. Controlling the release of a product and changes to it throughout the lifecycle by having controls in place that ensure consistent software via the creation of a baseline product.

Status Accounting. Recording and reporting the status of components and change requests, and gathering vital statistics about components in the product.

Audit and review. Validating the completeness of a product and maintaining consistency among the components by ensuring that the product is a well defined collection of components.

Depending on the intentions of the creator, software CM literature (cf., Conradi & Westfechtel, 1998) divides versions into three versioning dimensions (Estublier & Casallas, 1995). These dimensions should be fully orthogonal to each other.

Historical versioning. Versions that are created to supersede a specific version, e.g. for maintenance purposes, are called revisions. In practice, a revision of a component is usually created by modifying a copy of the most recent revision. The old revisions are permanently stored for maintenance and documenting purposes; they form the version history of the component.

Logical versioning. A variant is created as an alternative to a specific version. They are created in branches, that is, parallel development threads that may eventually be merged with the main development thread. Permanent variants are created when the product is adapted to different environments. Variance can again arise in several dimensions, including varying user requirements and varying system platforms, but also variants for testing and debugging.

Cooperative versioning. A temporary variant is a variant that will later be integrated (or merged) with another variant. Temporary variants are required, for instance, to change an old revision while the new revision is already under development.

2.4.2 Modelling aspect and versioning

A conceptual domain model is not developed all at once, but rather through a process of consecutive iterations (Solvberg & Kung, 1993). As model development becomes a more ubiquitous and collaborative process, support for model versioning becomes necessary and essential. This support must enable users to compare versions of model and analyze differences between them. Furthermore, as models become larger, collaborative development of models becomes more and more common. Model developers working in parallel on the same model need to maintain and compare different versions, to examine the changes that others have performed, and to accept or reject the changes. In fact, this process is exactly how developers collaborate on editing software code and text documents.

Different levels of abstraction of the same system enable to deal with complexity by removing details from model. The model must be able to act as an efficient and effective communications medium between the different parties involved in development project. Usually, models are augmented by details on each development step.

The success of distributed project depends on how well “laissez-faire” rule is obeyed, meaning that developers should be allowed to express what they want in whatever form. In a collaborative environment where different users work on models, it is important that there is a way of sharing own views, and step-by-step achieving agreement and common conceptualization.

Although, the initial goal is usually to develop a single model of the UoD, it turns out to be very important to preserve and model the various “views” of the information seen by different stakeholders and participants during the system analysis phase. Usually, different developers might have different vocabulary to express their perception of the world. It is important to preserve knowledge of developer that is expressed in the model fragment she has developed. We need to ensure that developer’s work will not be disturbed, for instance, if a developer uses term ‘aircraft’ referring to ‘airplane’, this term should be preserved in her local view, otherwise after several changes it will be difficult to continue.

In a model-versioning environment, given two versions of a model, users must be able to: (1) examine the changes between versions visually; (2) understand the potential effects of changes on applications; and (3) accept or reject changes done by colleagues.

The fields of software evolution and collaborative document processing have faced these challenges for many years. There is one crucial difference, however: in a case of software code and documents, what is usually compared - with only a few exceptions - are text files. For models, it is necessary to compare the structure and semantics of the models and not their textual serialization. Two models can be exactly the same conceptually, but have very different textual representations. For instance, their XML syntax may be different. The order in which definitions appear in the text file may be different. A representation language may have several mechanisms for expressing the same semantic structure. Thus, text-file comparison is largely useless for models.

Automatic comparison of versions. Given two versions of the same model, we must identify what has changed from one version to the next. This identification should be performed at conceptual level, that is, it should be expressed in terms of changes to model concepts, such as class concepts, and individual concepts, attributes and their values, relations and operations between concepts.

Contextual presentation of changes. If the user needs to understand or assess a change in model (e.g., deleted class), she should be able to see the change itself, but also to see its context. For instance, if a class was deleted, the user may want to know where in the class tree was the class located, whether it had any subclasses, what were its properties, and so far.

Navigation among changes. Changes often occur in different and unrelated places in model. Having examined one of the changes, the user must be able to navigate easily to the next change.

Access to old and new values. Understanding and assessing changes is impossible without ready access to both old and new values. Just knowing that the class name has changed and knowing its new name is not enough: when examining the change, we would like to know what the old value was.

2.4.3 View reconciliation

A system can be described from many viewpoints. Each viewpoint defines what characteristics should be included in its views and what issues should be ignored or treated as transparent. A view is, therefore, a piece of the model that is small enough to comprehend but that also contains all relevant information about a particular concern. Variants dimension is more concerned with different versions and configurations.

View reconciliation is an important for successful systems development in order to reach consensus in a social reality as discussed earlier. Since construction of a conceptual model of "reality" as it is perceived by someone is partly a process of externalisation of parts of this person's internal reality, and will in the first place act as organizational reality for the audience of the model. This model can then be used in the sense-making process by the other stakeholders, internalizing the views of the others if they are found appropriate. Despite any effort spent on externalising the thoughts of the stakeholders, misunderstanding will most likely happen.

Nevertheless, there is a need to reconcile these different perceptions in order to achieve a common conceptualisation and shared "social reality" in geographically distributed project. A conceptualisation mismatch is a difference in the way a domain is interpreted, whereas an explication mismatch is a difference in the way the conceptualisation is specified (Visser et al., 1998).

Conceptualisation mismatches are further divided into *model coverage* and *concept scope* (granularity).

- *Scope.* Two classes seem to represent the same concept, but do not have the same instances, although they may intersect. The classical example is the class "employee", where several administrations use slightly different concepts of employee, as mentioned by Wiederhold (1994).
- *Model coverage and granularity.* This is a mismatch in the part of the domain that is covered by the ontology, or the level of detail to which that domain is modelled. Chalupsky (2000) gives the example of an ontology about cars: one ontology might model cars but not trucks. Another one might represent trucks but only classify them into a few categories, while a third ontology might

make very fine grained distinctions between types of trucks based on their physical structure, weight, purpose, etc.

Further, explication mismatches are divided into *terminological*, *modelling style* and *encoding*.

- Two types of differences can be classified as terminological mismatches. A main problem is a human factor, i.e., use of different terms with the same intension (synonyms), or that we use the same term but with different intension (homonyms).
- Modelling style is related to the paradigm and conventions taken by the developers.
  - Paradigm. Different paradigms can be used to represent concepts such as time, action, plans, causality, propositional attitudes, etc. For example, one model might use temporal representations based on interval logic while another might use a representation based on point (Chalupsky, 2000).
  - Concept description. This type of differences are called modelling conventions in (Chalupsky, 2000). Several choices can be made for the modelling of concepts in the ontologies. For example, a distinction between two classes can be modelled using a qualifying attribute or by introducing separate class.
- One last mismatch in the explication category is encoding. Encoding mismatches are differences in value formats, like measuring distance in miles or in kilometres.

Noy and Musen (2000) define mapping as establishing correspondences among the models, and determining the set of overlapping concepts, concepts that are similar in meaning but have different names or structure, and concepts that are unique to each of the sources. Further, two relevant concepts: merging and alignment are also defined. Merging is to create a single coherent model that includes the information from all the sources. Alignment is to make the models consistent and coherent with one another but kept separately.

To avoid further confusion, thereinafter, we will use the following terminology when talking about view (model fragment) reconciliation:

- *Merging, integrating*. Creating a new model from two or more existing models with overlapping parts.
- *Aligning*. Bring two or more models into mutual agreement, making them consistent and coherent with each other.
- *Mapping*. Relating similar (according to some metric) concepts or relations from different models to each other by specifying the semantic similarity between them.

2.4.4 Notification

Change is permanent in a big scale development, in order to control consistency of the product fragments, the corresponding developers need to be informed about the actions of colleagues. Shen and Sun (2002) discuss selective notification as mechanism to be informed in big scale projects, usually triggered by users with explicit selection criteria. Selection criteria can be predefined, notification could also be automatically triggered by the system. These criteria are application dependent. Some usage examples are:

- Time criterion: a user can selectively propagate/accept updates made within a certain period of time.
- Object criterion: a user can selectively propagate/accept updates made to certain objects.
- Type criterion: a user can selectively propagate/accept certain types of updates.
- Version criterion: a user can selectively propagate/accept updates made in certain versions of a shared artefact.
- User criterion: a user can selectively notify certain users of her/his updates or selectively accept updates made by certain users.

In distributed environment it is difficult to keep overview on who is doing what. Therefore active and targeted information delivery is very important in this kind of development projects.

2.5 Requirements

Here we summarise the discussion in this chapter to a list of requirements that are important for distributed development. Farshchian (2001) has emphasized a list of requirements for product development environments to enable collaboration in geographically distributed developments. There we adopt certain requirements as listed below.

Req1. *Flexible access to the product.* A product development environment should provide flexible mechanisms for accessing and updating the product, i.e. no object locking.

Req2. *Unrestricted product fragment types.* A product development environment should allow the developers to share any type of development object that they might find useful for supporting their cooperation.

Req3. *Unrestricted relation types.* A product development environment should allow the developers to create any type of relation between any two fragments of product.

Req4. *Incremental product fragment refinement.* A product development environment should provide the developers with flexible mechanisms for incrementally refining the product. The developers should be allowed to start with vague products, and to refine them into more complete and formal ones.

Req5. *Support for boundary objects*. A product development environment should allow the developers to view the product from different perspectives. The environment should in addition support a global view of the product.

Req6. *Active delivery of information*. A product development environment should take an active part in delivering necessary information to the developers. In particular information about changes to the related product fragments should be delivered continuously to the interested stakeholders.

In addition to the above described requirements, as it was discussed earlier in this chapter, it is important that the following requirements are covered in distributed cooperative IS engineering centred on domain model:

Req7. *Knowledge externalisation in a means of conceptual domain model*. The means should be provided for stakeholders to externalise their perceptions of the problem domain in a form of a conceptual domain model.

Req8. *Domain concepts explanation (extension)*. Provide the means to explicate the concepts, i.e. definition in natural language should be provided. That will facilitate understanding the colleagues' views.

Req9. *Support for knowledge internalisation*. Provide the means to compare and align different views.

Req10. *Conceptual domain model should be available through whole development life-cycle*. The conceptual model should be easily accessible for all stakeholders as it represents common conceptualisation, e.g., describing the problem to be solved, and serves as common reference point for geographically dispersed project members.

Req11. *Flexible metadata specification about development objects*. It is important in a geographically dispersed development explicate the content of the product fragment in order to communicate the meaning. That should be done in the least labour-consuming way.

Req12. *Efficient dependency management*. The process of dependency relationships specification should be efficient, taking into consideration variety of object types, tools used.

2.6 Summary

The value of conceptual domain modelling is recognised in a multitude of settings, for different application areas. The key to achieving meaningful communication among stakeholders is getting them to share the relevant conceptual knowledge (Solvberg & Kung, 1993). This is usually achieved by developing conceptual models. Here we adopt a conceptual domain model as a means to facilitate communication among all stakeholders of the distributed system development project.

Therefore, in this chapter we have analysed the role of product in distributed IS engineering. We have shown the importance for stakeholders to share conceptual vision of product by externalising their perceptions, aligning them and finally committing to a common conceptual model (constructed “social reality”). We have shown that conceptual modelling is considered in most of existing development life-cycle methods as the important phase. Nevertheless, not so many developments methods are using the conceptual model throughout all phases. Finally, we have proposed a set of twelve requirements to facilitate management of distributed development, specifically, developed product fragments.

3
State-of-the-Art Surveys

"Although there is a bunch of appropriate techniques and powerful tools, none of them is sufficient for solving all involved problems."
— Axel Mahler, Variants

The goal of this chapter is to provide a background survey of the technological environments underlying the realisation of the method of this thesis, stating the relation to this work. To begin with, section 3.1 will give an overview of Computer Aided System Engineering (CASE) tools. Then we go through CASE tools ability to support cooperative IS engineering in distributed teams. Afterwards, in section 3.3 we take a look at Computer Supported Collaborative Work (CSCW) tools. Next section overviews state-of-the-art in the realm of repository systems including content management systems and their support for IS engineering. Then we take a closer look at different techniques used to manage diversity of the development objects in distributed IS engineering. The following techniques are considered relevant: integrated modelling languages, traceability, view alignment.

Tools are reviewed considering the list of twelve requirements stated in Chapter 2. Summary of the chapter concludes the state-of-the-art survey by showing how well these techniques support distributed IS engineering.

3.1 CASE Tools and Modelling Frameworks

The terms *Software Engineering Environments* (SEEs) and *Computer Aided Software/Systems Engineering* (CASE) denote tools and groups of related tools that are used to support the work needed to develop a software product. Tools are the building blocks in CASE and SEE, and range from assemblers and compilers, to graphical editors for creating visual diagrams, to project and workflow management tools. Tools are often grouped according to various criteria, such as the project phase they are used in, the type of activities they support, the information in form of development objects they exchange, the target user group, etc. Different types of CASE and SEE may support different aspects of work, e.g. creation and sharing of development objects, definition of work processes and methods, etc.

There exist different definitions of these terms in the literature. In terms of Sommerville (1992) SEEs are regarded as consisting of *CASE building blocks*, where these building blocks are *integrated* in different forms of *environments*. Sommerville also distinguishes between the types of environments that can be created, based on the type of activities they support: *Programming environments* (for coding activities), *CASE workbenches* (for analysis and design activities), and *Software-engineering environments* (for whole life cycle).

A more comprehensive classification is provided by Fuggetta (1993). This classification distinguishes between a production process and a metaprocess. The production process includes "all the activities, rules, methodologies, organizational structures, and tools used to conceive, design, develop, deliver and maintain a software product" (Fuggetta,1993). A production process is "defined, assessed, and evolved through a systematic and continuing metaprocess" (Fuggetta, 1993). These two processes are supported by an infrastructure (a combination of operating systems, advanced databases, process technology, etc.), which is implemented using the enabling technology (standards that allow tools to be physically distributed and still cooperate with each other, e.g. integration platforms such as network file systems). The infrastructure, production process support, and metaprocess support together constitute the software process support.

Fuggetta's definition of CASE is more generic than that of Sommerville's (1992). In Fuggetta's terms, CASE is considered to be any combination of enabling technologies and software process support technologies. This means that CASE may support both the production process and the metaprocess. Fuggetta further classifies CASE used in the production process as: CASE tools (used to support single tasks), CASE workbenches (used to support activities consisting of tasks) and CASE environments (used to support a possibly large part of the process).

CASE technology is seen as "an interoperable, computerised tool set designed to support stakeholder tasks and processes over the full information systems development lifecycle" (Lundell & Lings, 2004).

So, a CASE tool is a computer-based product aimed at supporting one or more techniques within a software development method (such as a structured or object-oriented method). CASE tools have normally been closely associated with the notations and procedural practices of specific design methods. Such tools are therefore to likely have user interfaces that are either influenced strongly by the method and its preferred strategy or, much worse, by the internal structures used to store the design model (Budgen et al., 1993). There are CASE products that address multiple methods and Meta-CASE products that generate CASE tools from method specifications (Jarzabek & Huang, 1998). Upper-CASE tools focus on the system analysis and logical design phases, while Lower-CASE tools focus on the construction of software systems. CASE tools can be integrated together to form a more sophisticated CASE environment. This can be done in several dimensions, such as presentation integration, control integration, process integration, and data integration (Chen & Norman, 1992), as shown in Figure 3.1.

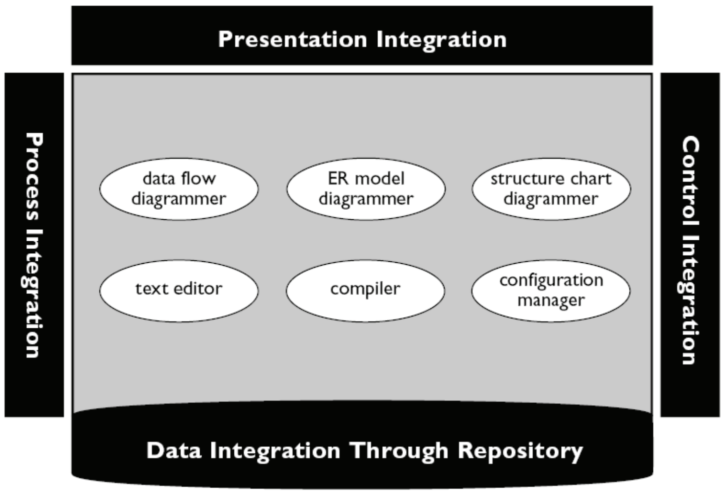


Figure 3.1 Dimensions of CASE tools integration (Chen & Norman, 1992)

A SEE is a computerised system that provides support for the construction, management and maintenance of a software product (Brown et al., 1992). A SEE consists in a repository that stores all the information related to the software project throughout its life cycle, and tools that support the involved technical and managerial activities. SEEs differ from one another depending on their database nature, scope of provided tools or adopted technology.

3.2 Collaborative CASE tools

CASE is a mature technology and has existed since the middle of 70's, though cooperation support is still quite limited in existing CASE tools. More and more product development projects are being conducted by geographically dispersed groups. Stepwise advancements in cooperative technologies, in particular those originating from the CSCW research field, are influencing CASE research and development to a degree that one would expect. If CASE tools cannot support the cooperation among these groups, the use of CASE will be marginalized. Especially, as many CASE tools are developed as "time sharing" systems

(Farshchian, 2001): each developer is given the feeling of being the “only user” of the system.

In addition, the format of the product objects are often controlled by strict consistency checks, making the evolution of product objects from informal ideas to formal constructs difficult. As a consequence, CASE tools might be reduced to tools for documenting products that are developed outside the CASE tools. That is not what CASE tools are built for. CASE tools in the best case confine themselves to offering a central repository where information about the product can be accessed regardless of geographical location.

Weaknesses in CASE tool support could be divided into the following aspects (Kelly et al., 1996) as follows.

- Lack of mechanism for integrating sets of methods while maintaining consistency between various models,
- Lack of support of multiple users to create, modify and delete sets of partly overlapping model instances,
- Inadequate catering for multiple representational requirements raging from fully diagrammatic to fully textual or matrix representation.
- Failure to provide consistent mapping mechanism between different representational paradigms.
- Lack of flexibility and evolvability in method support ranging from syntactic variation in methods to crafting totally new method components.
- Insufficient catering for different information-related needs of a diverse set of stakeholders.

CASE tools are supposed to increase productivity, improve the software product quality and facilitate Information Systems development (Jarzabek & Huang, 1998). However, they have been failing to deliver the benefits they promise (Iivari, 1996). Previous research (Kemerer, 1992) reports that one year after introduction, 70% of the CASE tools are never used, 25% are used only by a limited number of people in the organization, and 5% are widely used but not to capacity.

According to a study by the Standish Group (1994), only 12% of software development projects are completed on-time and within budget. The average cost overrun is 189% over their initial budget estimate, is completed 222% over original time estimate, and incorporates only 61% of originally specified features and functions.

The collaborative CASE tools should keep track of changes in different working modes (Lee et al., 2001) as follows.

- Multiple developers are working on a single common version simultaneously;
- Developers are working individually on their local versions;
- Both cases: some developers collaborating synchronously, others working individually.

In first case the system should track and resolve multiple edits by different developers in the same fragment. Changes could occur simultaneously or sequentially in collaborative session. Considering second case – there will be a number of current versions – the system should provide and manage awareness of their existence and dependencies among them. Third case incorporates previous both – asynchronous and synchronous developing.

3.2.1 Rational Rose

IBM Rational Rose® (IBM, 2005a) is one of the most advanced CASE tools. One of its strengths is open architecture and possibility to integrate other tools, or "home-made" various plug-ins. For instance, included Eclipse IDE allows connecting multi-functional Eclipse open-source development platform (see section 3.2.4). Enterprise Edition has integration with Microsoft® Visual Studio™ and other Java™ platform IDEs.

Rational Rose supports UML modelling language and model-driven development is supported. A useful feature in distributed development is a free-form diagramming, which may help explaining own ideas to remote colleagues in an informal way. That feature satisfies *Req2*, *Req3* and partially *Req4*, but all these requirements are satisfied only for the modelling phase of development. As a modelling tool, Rational Rose satisfies *Req7* and *Req9*, though it might be questionable to what extent UML is useful for conceptual problem modelling.

3.2.2 Medius Visual Ontology Modeler

Visual Ontology Modeler™ (Sandpiper Software, 2005) is one of the third party's add-in to Rational Rose. It is a UML-based ontology modelling tool that enables component-based ontology development and management for use in interoperability solutions. Main features of the tool are as follows.

- A multi-user, network-based environment for ontology development in a graphical notation;
- A set of ontology authoring wizards that create and maintain the required UML model elements for the user, reducing construction errors and inconsistencies;
- Export facilities in XML schema, RDF, and other formats.

The Visual Ontology Modeler implements Sandpiper's UML Profile for Knowledge Representation (Sandpiper Software, 2005), which extends UML to enable modelling of knowledge representation concepts such as class, relation, function. It also includes a library of ontologies that represent the IEEE Standard Upper Ontology (SUO), concepts relevant to XML schema, and RDF generation, as well as other basic concepts required to develop ontologies, i.e. for the commerce and bioinformatics. Therefore, the tool does satisfy *Req7* and *Req9*.

3.2.3 LibreSource

LibreSource (Libresource, 2005) is an open free software platform that aims at hosting virtual teams and distributed communities for different activities, including co-authoring, co-development and co-engineering activities, for software and non-software applications. It is similar to SourceForge[4], well known open source community service. LibreSource is accessible through a web portal and its main aim is to provide an alternative to cooperative development platforms such as SourceForge/CVS, and community management.

The search engine indexes all the data hosted in the platform, as well as uploaded files (Word, PDF, Open Office). LibreSource has a naming scheme that allows to access documents and resources through tree-like hierarchy. LibreSource enables to easily and quickly define public and private areas in respect with different users involved. This database supports the automatic archiving of online data. Main features of the platform are as follows.

- Coordination is based on a cooperative workflow system, compliant to WfMC specifications.
- Awareness is supported by an event manager that is coupled with all the components of LibreSource.
- Communication is supported by common groupware tools as forums, bug trackers, etc.

The aim of LibreSource is to facilitate the cooperative software development and the management of geographically spread teams having to work together on a common project shared on the Internet. Mainly, requirements *Req1*, *Req2*, and *Req6* are satisfied.

3.2.4 Eclipse platform

Eclipse platform (Eclipse, 2005) is a famous open-source package of commercial tool quality. Open architecture is the best feature, allowing to create various plug-ins and extend Eclipse by the desired functions. In the following, we overview couple plug-ins supporting collaboration. The major Eclipse Platform Components are shown in Figure 3.2 and consists from the following components:

- Platform Runtime;
- Workbench which implements the graphical interface to Eclipse;
- Workspace that “holds” the development environment;
- Version and Configuration Management (VCM) system.

Eclipse platform has become widely used because of its open architecture. Consequently, there are plenty of various plugins implemented for Eclipse. Below two plugins concerning collaborative work are shortly overviewed.

[4] http://sourceforge.net

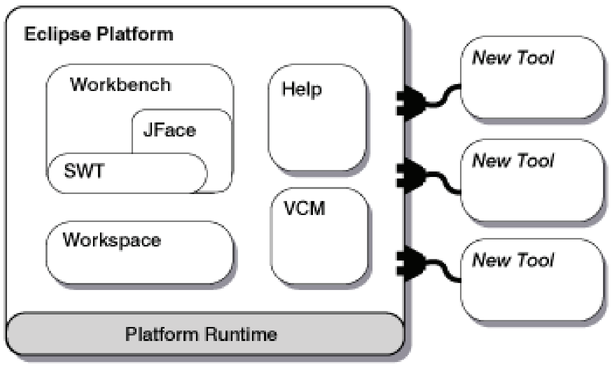


Figure 3.2 The Eclipse Platform architecture

Composent Eclipse Plugin

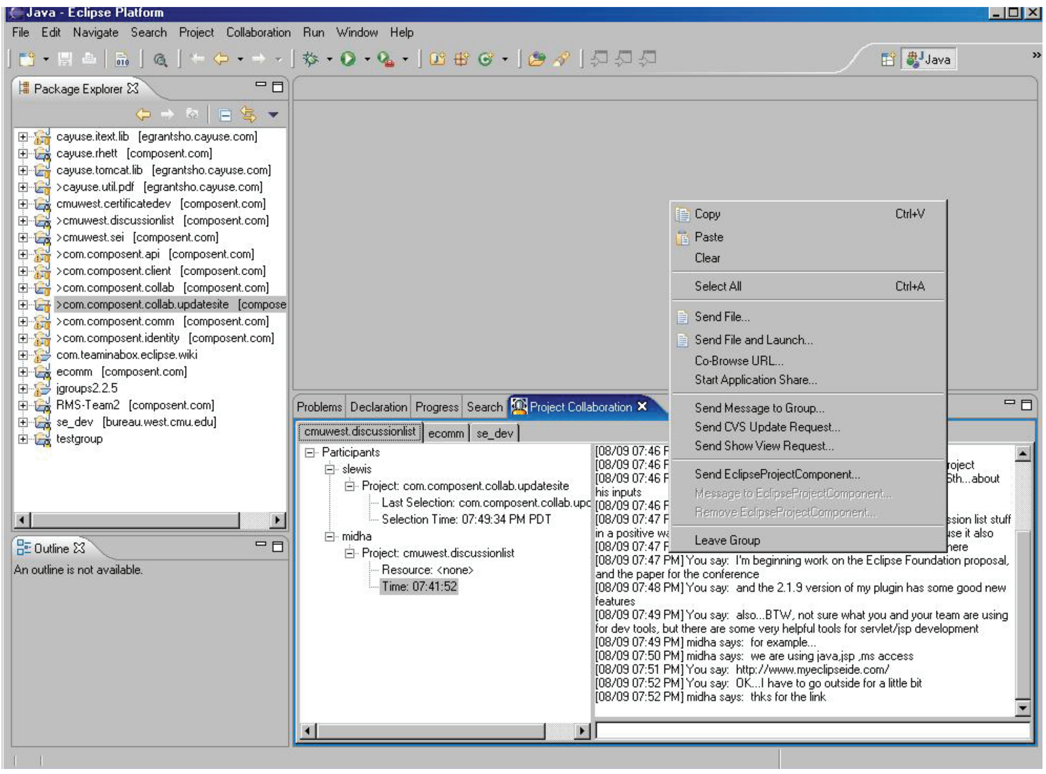
This plug-in (see screenshot in Figure 3.3) provides secure, real-time collaboration associated with Eclipse projects (Composent, 2005). The main user collaboration features, most relevant to the settings described in chapter 2 of this thesis, are as follows.

- Version control is integrated with CVS source code management;
- Awareness for project teams. Members receive presence information, and also information from other Eclipse users about their current activities (tasks, open editors, current UI selection). The system is extensible to allow other sorts of team-specific presence information to be automatically communicated among team members;
- File Sharing enables team members to transfer files to one another;
- Messaging and Chatting provides direct communication tools for team members. They can send IM's (Instant Messages) and alerts to one/all other group members.

Sobalipse - an Eclipse plugin

Sobalipse is another Eclipse plugin, which implements real-time collaboration. Sobalipse enables to work remote users, such as, tele-pair programming, real-time code reviewing, etc. Sobalipse (2005) is a framework – it is possible to develop Eclipse plugins for real-time collaboration with Sobalipse plugin's development API.

In general, Eclipse platform is an extensible application that might be turned to really powerful CASE tool. Though, it is mainly programming environment with a help of plugins the tool can span whole life-cycle of systems development. As reviewed above, there are already implemented plugins to facilitate collaboration. There are written plugins for modelling as well. By now we would characterise it as satisfying requirement *Req6*, and partially, *Req1*.


Java - Eclipse Platform
File Edit Navigate Search Project Collaboration Run Window Help
Java
Package Explorer
cayuse.itext.lib [egrantsho.cayuse.com]
cayuse.rhett [composent.com]
cayuse.tomcat.lib [egrantsho.cayuse.com]
>cayuse.util.pdf [egrantsho.cayuse.com]
cmuwest.certificatedev [composent.com]
>cmuwest.discussionlist [composent.com]
>cmuwest.sei [composent.com]
>com.composent.api [composent.com]
>com.composent.client [composent.com]
>com.composent.collab [composent.com]
>com.composent.collab.updatesite [compose
>com.composent.comm [composent.com]
>com.composent.identity [composent.com]
com.teaminabox.eclipse.wiki
ecomm [composent.com]
jgroups2.2.5
RMS-Team2 [composent.com]
se_dev [bureau.west.cmu.edu]
testgroup
Outline
An outline is not available.
Copy Ctrl+V
Paste
Clear
Select All Ctrl+A
Send File...
Send File and Launch...
Co-Browse URL...
Start Application Share...
Send Message to Group...
Send CVS Update Request...
Send Show View Request...
Send EclipseProjectComponent...
Message to EclipseProjectComponent...
Remove EclipseProjectComponent...
Leave Group
Problems Declaration Progress Search Project Collaboration
cmuwest.discussionlist ecomm se_dev

```
Participants
  slewis
    Project: com.composent.collab.updatesite
      Last Selection: com.composent.collab.upc
      Selection Time: 07:49:34 PM PDT
  midha
    Project: cmuwest.discussionlist
      Resource: <none>
      Time: 07:41:52
```

```
[08/09 07:46 F
[08/09 07:46 F                                        roject
[08/09 07:46 F                                        5th...about
his inputs
[08/09 07:46 F
[08/09 07:47 F                                        ssion list stuff
in a positive wa                                      use it also
[08/09 07:47 F                                        here
[08/09 07:47 PM] You say:  I'm beginning work on the Eclipse Foundation proposal,
and the paper for the conference
[08/09 07:48 PM] You say:  and the 2.1.9 version of my plugin has some good new
features
[08/09 07:49 PM] You say:  also...BTW, not sure what you and your team are using
for dev tools, but there are some very helpful tools for servlet/jsp development
[08/09 07:49 PM] midha says:  for example...
[08/09 07:50 PM] midha says:  we are using java,jsp ,ms access
[08/09 07:51 PM] You say:  http://www.myeclipseide.com/
[08/09 07:52 PM] You say:  OK...I have to go outside for a little bit
[08/09 07:52 PM] midha says:  thks for the link
```

Figure 3.3 Screenshot of Composent Eclipse plugin

3.2.5 Other CASE solutions

Here we shortly discuss some recent research initiatives resulted in implemented prototypes.

VRCASE

VRCASE (Bu et al., 2001) is a virtual environment based CASE tool. It provides a 3D multi-user collaborative software modelling environment with automatic object-class abstraction, class diagram generation, and C++ skeleton generation facilities for assisting Object-Oriented software development. It allows multiple concurrent users to model software system collaboratively. To achieve efficient collaborative software development, VRCASE has implemented a fine-grained locking and notification mechanism together with visual indicators to maintain system consistency among multiple concurrent users. Therefore, the tool satisfies *Req5* and partially both, *Req1* and *Req4*.

DOSDE

Oliveira et al. (2004) define the concept of “Domain-Oriented Software Development Environment” (DOSDE). This kind of environment readies knowledge about a specific domain in a symbolic representation (a domain ontology). It also considers a library of potential tasks from the domain to support problem understanding. This approach targets *Req7*, *Req8*, and *Req9*.

Software Design Board

Wu and Graham (2005) propose a tool called the Software Design Board, a collaborative design tool that supports a variety of styles of collaboration and facilitates transitions between asynchronous and synchronous styles of collaboration, and between co-located and distributed styles of collaboration. The whiteboard space can be divided into any number of segments, which allow data to be shared in different ways. As software engineers work together in a variety of styles and move frequently between these styles throughout the course of their work. Consequently, requirements *Req2*, *Req3*, *Req4*, *Req5* and *Req6* seem to be satisfied, in addition to partially satisfied *Req1*.

3.3 Computer Supported Cooperative Work

How do stakeholders manage to overcome the barriers to coordination that are imposed by distance? How do distributed developers maintain group awareness? All these questions were posed and analysed by Farshchian (2001).

Group awareness information includes knowledge about who is in a project, where in a code they are working, what they are doing, and what their plans are. This knowledge seems vital if distributed developers are to coordinate their efforts, smoothly add code, make changes that affect other modules, and avoid rework.

Collaborative software, also known as *groupware*, is application software that integrates work on a single project by several concurrent users at separated workstations.

According to Carstensen and Schmidt (2002), CSCW addresses "how collaborative activities and their coordination can be supported by means of computer systems". On the one hand, many authors consider that CSCW and groupware are synonyms. Ellis et al. (1991) define groupware as "computer-based systems that support groups of people engaged in a common task (or goal) and that provide an interface to a shared environment". On the other hand, different authors claim that while groupware refers to real computer-based systems, CSCW focuses on the study of tools and techniques of groupware as well as their psychological, social, and organizational effects. The definition of Wilson (1991) expresses the difference between these two concepts:

> *"CSCW [is] a generic term, which combines the understanding of the way people work in groups with the enabling technologies of computer networking, and associated hardware, software, services and techniques."*

There are many different CSCW tools targeting to support collaboration and cooperation in different areas, though there is no CSCW tool that would be particularly created to support systems development activity. Of course, there are attempts to incorporate CSCW technology into CASE tools, but these attempts are restricted to support multi-user access, for instance some of the earlier discussed tools.

Collaborative Editing

Very interesting trend is to adopting existing tools for collaboration purposes without changing them. The biggest advantage of such approach is that users are used to the tools, they do not need to learn or adjust to new tools.

CoWord (Xia et al., 2004) is one of the first tools that allow multiple users to edit the same Microsoft® Word™ document at the same time over the Internet. CoWord retains the "look-and-feel" and functionalities of MS Word. CoWord supports unconstrained editing style, giving users complete freedom in their way of using CoWord to support individual and group work.

For collaborative system designers, CoWord demonstrates an innovative technology that integrates state-of-the-art collaborative editing techniques such as REDUCE (REal-time Distributed Unconstrained Collaborative Environment) and GRACE (GRAphics Collaborative Editing) with off-the-shelf single-user editors in a collaboration-transparent way (Reduce, 2005), i.e., *without changing the source code of the existing single-user application*. The GRACE project aims to research develop, and apply innovative technologies for supporting collaborative graphics editing. The scope of the GRACE project includes both object-based and bitmap-based collaborative editing systems. Object-based GRACE systems can be used as multi-user CAD/CASE applications. Bitmap-based GRACE systems can be used as multi-user drawing tools or as electronic whiteboard systems. Major collaborative word processing features of CoWord Demo are as follows.

- Real-time concurrent editing of objects of any type (i.e., MS Word objects, for instance, formatted texts, graphic objects, clip-art objects, tables, bulleting and numbering, paragraph alignment, etc.) at any granularity (down to individual text characters, graphic lines, etc.) in any part of the same document;
- Undo operations on objects of any type at any times in multi-user environments;
- Concurrent editing and commenting on the same document;
- Concurrent tracking and automatic merging of modifications in multi-user environments;
- Detailed workspace awareness support.

In general, CSCW tools provide good support for functionality covered by requirements *Req6*, and can be used to satisfy demand for *Req7*, *Req8*, *Req9*. Furthermore, these tools focus on collaboration around some objects; therefore *Req1* is typically well satisfied.

3.4 Repositories in IS Engineering

CASE repositories try to provide mechanisms for viewing a product object or a group of product objects from different perspectives, depending on the context and the background knowledge of the particular developer group. Mechanisms such as transaction control, version control, and concurrency control try to prevent one developer from destroying the result of the work done by another. Process integration efforts acknowledge the fact that a software process is a complex artefact that involves many activities and affects many developers. An integrated software process tries to bring together developers and their data in a way that the coordinated efforts of single developers can result in the large product.

Bernstein and Dayal (1994) review the meaning of the most important services offered by repository system and list most important features as follows.

- *Check-out/Check-in* allows users to copy the object of interest into their private workspace, i.e., check-out. After editing the object is checked in. This kind of development transactions is particularly useful for long activities, i.e., days, weeks.
- *Version Management* is responsible for keeping a version history of objects. Versions are meaningful and consistent snapshots of an object during its lifecycle. These versions are represented as objects in the repository and can be identified, retrieved, and compared to each other.
- *Configuration Management* allows managing collections of related objects. Versioned objects are sometimes associated with versions of other objects. This binding between versions of different objects defines a configuration. CM should have appropriate mechanisms to identify and represent these configurations. Moreover, methods for linking and de-linking objects as well as consistent change-propagation methods are mandatory.
- *Context Management* limits the view of developers to specific objects which are necessary for a particular task, the context. Contexts define a semantical view on the content of a repository, in contrast to configurations which give a view on the physical interrelations. Like them context management should offer mechanisms to define, identify and represent contexts.
- *Notification* is the ability to inform developers of certain events. For instance, changes of objects can cause notifications to developers working on the same or related objects.
- *Workflow Control* considers the phases through which objects progress during their lifecycle. In system development, for instance, an object goes through requirements, design, implementation, testing, and documentation phases. Objects should be assigned to any of these phases according to their lifecycle. When they evolve, promote and demote operations advance the state or go back to a previous one. This can be initiated manually by the developer or automatically by notification rules.

3.4.1 Metis Team Server

Metis enterprise modelling environment (Metis, 2005) is supported by server-based model repository, which allows models to be stored and shared in a central model repository. Repository items are organised in, and navigated through, a standard, hierarchical folder structure.

All models are versioned. A model's full version history is stored in the repository. Versions may be retrieved using version policies such as newest version in production. Models in the repository can only be changed after checking them out, and can only be made available to other users by checking them back in. Each check-out/check-in creates a new version. Checking-out a model gives a write-exclusive ("pessimistic") lock, so that only one user may change a model at a time. This locking mechanism does not satisfy requirement for flexible access to product fragments.

Each sub-model (model fragment) is stored in the repository as a separate item with its own version history/check-outs/check-ins. This feature allows model dependency tracking, i.e., after splitting a model into separate files (sub-models), the Team Server tracks file dependencies. It allows to set up version policies specifying which version of each dependent file to include.

The Metis Team Server repository can store any digital content, not just Metis models. Versioning, check-out/check-in, export and import are available for any file. Using Metis Model Annotator (a modelling environment) it is possible to annotate models in the repository. Annotation models can be stored in the repository as well.

Tool allows defining various types of relation and creating new type of objects, i.e., satisfies requirements *Req2* and *Req3*. As well requirements *Req4* and *Req5* are partially satisfied. Obviously *Req1* is not satisfied because of locking enforced in storage. Though, after merging with Troux Technology, Metis repository was replaced by Troux Object Repository with much finer-grained control of stored objects.

3.4.2 Unicorn Workbench

The Unicorn Universal Repository (Unicorn, 2005) provides enterprise-scale storage and management for technical metadata and for enterprise architecture objects, including the Ontology Model and Semantic Mappings. It uses an OMG MOF-based flexible metamodel to ensure that all types of technical metadata and enterprise architecture objects are stored and cross-referenced. Dublin Core standard is used to document metadata. The Unicorn Universal Repository provides an environment with permissions management, multi-user collaboration, and versioning. Features are as follows.

- Capture common business language in an ontology model;
- Provide business semantics (metadata) to multiple data assets;
- Re-use and extend existing ERD & UML models;

- Automatically import off-the-shelf industry models;
- Test instances to demonstrate and validate the ontology model and business rules.

The workbench satisfies requirements *Req7*, *Req9*, *Req11*, partially *Req8*. Unfortunately, it is not so clear how well requirements *Req1* and *Req12* are satisfied by the tools workbench.

3.4.3 Microsoft Meta Data Services

Microsoft has launched Meta Data Services (previously known as Microsoft Repository) as part of Microsoft® SQL Server™ (Microsoft, 2005). The repository supports modelling activity, i.e. by providing model storage. Actually any modelling language can be supported as schema for model storage is created based on XMI definition of a particular modelling language, called the Open Information Model (OIM). OIM is a formal specification of metadata that provides common ground for defining standard metadata. To achieve maximum integration across its product lines, Microsoft uses the OIM standard when defining metadata constructs. Figure 3.4 illustrates basic interaction with Microsoft Repository.

Consequently, requirements *Req2*, *Req3* are satisfied (for modelling stage), as well as *Req6*, and *Req9*. However requirement *Req1* is not, because of partial access of the object while being edited.

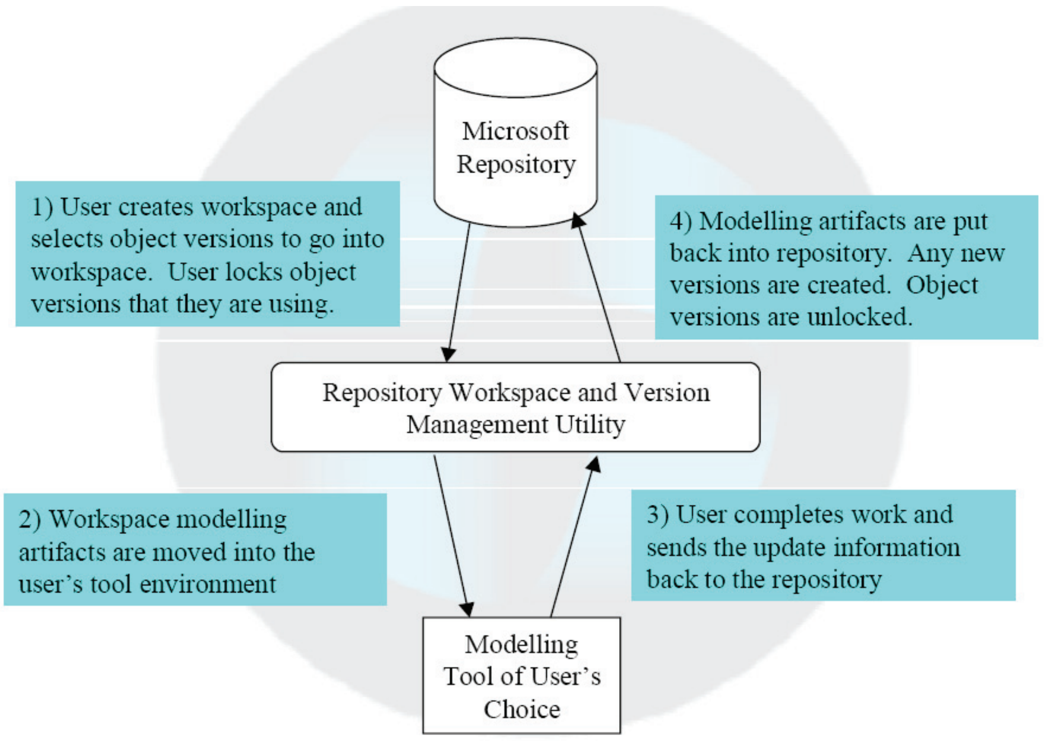

Microsoft Repository
1) User creates workspace and selects object versions to go into workspace. User locks object versions that they are using.
4) Modelling artifacts are put back into repository. Any new versions are created. Object versions are unlocked.
Repository Workspace and Version Management Utility
2) Workspace modelling artifacts are moved into the user's tool environment
3) User completes work and sends the update information back to the repository
Modelling Tool of User's Choice

Figure 3.4 Basic interaction with Microsoft Repository

3.4.4 Content Management Tools

A content management system (CMS) is a system used to organize and facilitate collaborative creation of documents and other content. A CMS is frequently a web application used for managing websites and web content, though in many cases, content management systems require special client software for editing and constructing documents.

CMSs allow users to provide new content in the form of documents. The documents are typically entered as plain text, perhaps with markup to indicate where other objects (e.g., pictures) should be placed. The system then uses rules to style a document, separating the display from the content, which has a number of advantages when trying to get many documents to conform to a consistent "look and feel". The system then adds the documents to a larger collection for publishing. The systems also often include some sort of concept of the workflow for the target users, which defines how the new content is to be routed around the system.

Enterprise CMS (ECMS) vary in their functionality. Some support both the Web and publications content life cycle, while others support the web content life cycle and either transactional content or customer relationship management content. ECMS usually contains components like document management, collaboration, business process management, records management, email management, workflow and web content management.

Next we overview openShore - an open source content management system, being one of the most advanced in its class.

OpenShore

The core system of OpenSHORE (2005) is not an application that can be used directly by an end user. It is a tool that processes XML documents, extracts information (objects and their relations) from these documents, stores them in the repository and makes them accessible through a web browser interface. A user can access the files directly using a Web browser, and open the project files of choice. Browsing between files can be done through hyperlinks and the native interface for that Web Browser. OpenSHORE has in addition a command line client implemented in Java that can be used to add documents to the repository or update them. The command line client can be used for integration with other tools like source code repositories and search for information.

As well as the documents, the repository stores a metamodel which describes the documents. The repository of OpenSHORE can be configured to store any kind of objects (e.g., system requirements, packages, interfaces, classes) with their relations (e.g., class implements interface). However, it requires that these documents are stored in an XML document, preferably XHTML. Documents that are not of this type need to be parsed and translated. Relations can then either be constructed automatically depending on the parser or done manually.

CMS obviously are not substitution for CASE tools, but can be used as supportive technology, i.e. mainly for storing and managing the development objects. CMS satisfies the following requirements: *Req2*, *Req3*, *Req11*, and *Req12*.

3.5 Information integration and management

In this subsection we discuss methods and tools dealing with information integration and management. Focus here is traceability methods (see section 3.5.3), i.e. means to discover and maintain the dependency links between the development objects. But first, we revisit use of ontologies in systems development, and then briefly discuss "unification approaches" in section 3.5.2.

3.5.1 Use of ontologies

An on-going research project (Saeki, 2004) is looking at supporting software-requirements elicitation and composing software from re-usable architectures, frameworks, components and software packages. They are developing relevant techniques through the use of ontology and its reasoning mechanism, to maintain semantic consistency. Ontology system (Saeki, 2004) has two layers; one for requirements elicitation and the other for re-usable parts. By establishing relationships between the two layers, the ontology system can play a role in bridging gaps between a requirements specification and an architectural design at a semantic level (Saeki, 2004).

(CEEBI, 2004) addresses the issue of collaborative multi-site distributed software development environments by dividing it into research issues as follows.

- Define the concepts, requirements, and representation of an ontology for multi-site distributed software development;
- Define an ontology-based software development architecture which addresses the needs of the collaborative, multi-site and distributed environment. This will address issues such as awareness, access control, security, communication and group decision support.

Ontology is the term used to refer to a conceptualisation of some domain of interest, which may be used as unifying framework to solve the problems. A key feature of an ontology is agreement about shared conceptualisations. The use of an ontology reduces conceptual and terminological confusion by providing a unifying framework within an organisation or community of users. In this way, an ontology enables shared understanding and communication between the front-end groups, backend groups and the application groups. In other words, different groups producing their own databases and residing at different sites can share a conceptual model.

Though these two approaches are undergoing research programs at their early stage, they focus on use of ontologies to capture semantics of development objects. Therefore, they target to support requirements *Req7*, *Req9*, *Req10*, *Req11*, and *Req12*.

3.5.2 Language families

One way to solve the problem of managing heterogeneous development objects is to adopt a language family for specifying all necessary development aspect. This would allow using similar notation, stepwise increasing level of details, for instance, UML notation, but preferably with more seamless transition between different diagrams.

RM-ODP

RM-ODP (2005) considers lifecycle of distributed systems from enterprise, information, computational, engineering and technology viewpoints. A viewpoint defines a set of related concerns that are important in the design of a system. A model defined from a particular viewpoint focuses on the particular concerns defined by the viewpoint. Viewpoints should be chosen with respect to requirements that are of concern to some particular group involved in the design process.

The use of different viewpoints in order to describe a system raises the issue of consistency, despite of being in the same language family. Descriptions of the same or related entities appear in different viewpoints. Therefore, it is necessary to assure that these multiple models are not in conflict with each other.

Enterprise modelling

Gustas and Gustiene (2003) propose three levels of information system models are necessary for maintenance of a systematic change, e.g., in order to understand why a technical system component is useful and how it fits into the overall organisational system. These levels are as follows.

- The pragmatic level;
- The semantic level;
- The syntactic level.

The most abstract is the strategy-oriented business process analysis level, which is referred to as pragmatic level. Strategic models are useful for illustration of the actual architectural solutions and general communication infra structure. They are necessary to provide motivation behind new business solutions that can be expressed in qualitative and quantitative terms. The semantic level must have a capacity to describe clearly the static and dynamic structures of business processes across organisation and technical system boundaries. The syntactic level should define implementation-oriented details, which explain the data processing needs of a specific application. Each level increases level of details, introducing additional new notational symbols, still keeping levels tied.

Language families are useful technique to provide a solution for requirements *Req7*, *Req9*, *Req10*, and *Req12*. However, it is not likely that all stakeholders in a geographically distributed project will be acquainted with a particular (software specification) language. While educating and training all members to use one particular language takes time, though future projects may benefit of it.

3.5.3 Traceability and product fragment management

The research area of requirements traceability has attracted a lot of attention from both practitioners and academic researches in the last two decades. Researchers tackle the problems of applied traceability by proposing various ways how to make requirements and other artefacts traceable. In this subsection we take a closer look at the management of product fragments dependency. Main focus here is on product traceability, but not on process traceability. Meaning that we survey approaches that deal with relating different product fragments, but not recording who and when produced a new revision of the product fragment, i.e. tracing the evolution of a particular product fragment.

Reference model for traceability

Ramesh and Jarke (2001) have conducted empirical studies in a range of software development companies and proposed reference model for the objects and traceability links to be recorded. A simple meta-model (see Figure 3.5) has been derived as part of their studies.

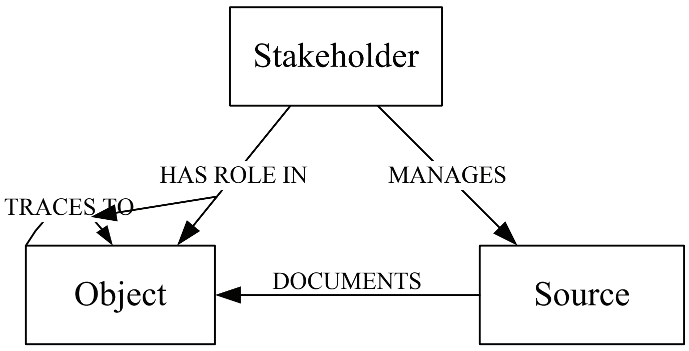


Figure 3.5 Traceability meta-model

Four general types of requirements traceability links were identified as an integral part of reference models (see Table 3.1).

Table 3.1 Traceability links types used in reference models

Link type	**Purpose**	**Uses**
Satisfaction links	To ensure that the requirements are satisfied by the system	- To ensure consistency between outputs of different phases of the lifecycle; - Trace the designs created to satisfy requirements; - Trace system/subsystem components to which requirements are allocated.
Evolution links	Document the input-output relationship of actions leading from existing objects to new or modified objects	Identify where the various objects come from. Identify the origins of various objects to facilitate: - Better understanding of requirements (or other objects); - Establishment of accountability of creation and modification of objects; - Tracking the modification, refinement history of various objects.
Rationale links	Represent the rationale behind the objects or document the reason for changes	Identify the reasons behind the creation of various objects and their modification, including: - Justifications for creation or modification; - Decisions and assumptions made; - Context in which the object were changed; - Transparency into the decision process including discarded alternatives.
Dependency links	Help manage dependency among objects (typically at the same stage of development), often imposed by a constraint.	Track the composition and hierarchies of objects and manage notification of changes in interdependent objects.

A Comprehensive Traceability Model

Toranzo and Castro (1999) broaden out a definition of requirements traceability: “Requirements traceability is the ability to describe and follow the life of a requirement, in both forward and backward direction, within the context of three composite, interrelated and parallels layers: organization/environment rules, management and development”. The layers are specified as follows.

- *Organizations/Environment rules layer*. This layer holds all the elements (goals, strategy, rationale, constraints, quality assurance, and change management policies) that exist prior to the creation of the project. These elements constrain partially or totally the development tasks.
- *Management layer.* This layer is subordinate to Organizations/Environment rules layer as project managers are responsible for ensuring that the software development complies with the organizations policies, goals and requirements. This layer holds elements (task, resource, milestone and risk) that should be taken into account.
- *Development layer.* This layer includes the definitions of pre and post-traceability. Pre-traceability is concerned with those aspects of a requirements life prior to inclusion in the requirements specification. Post-traceability deals with those aspects of a requirements life after its inclusion in a requirements specification.

Toranzo and Castro (1999) identify and differentiate elements used by different stakeholders, by that they establish relationships between view points of different users, particularly project manager, requirement engineer and software engineer.

A framework for requirements traceability based on UML

Letelier (2002) presents a framework for configuring requirements traceability by integrating textual specifications and UML model elements. Proposed approach is restricted to UML language and can be applied to software process based on UML. In Figure 3.6 meta-model of approach for requirements traceability is presented by means of class diagram.

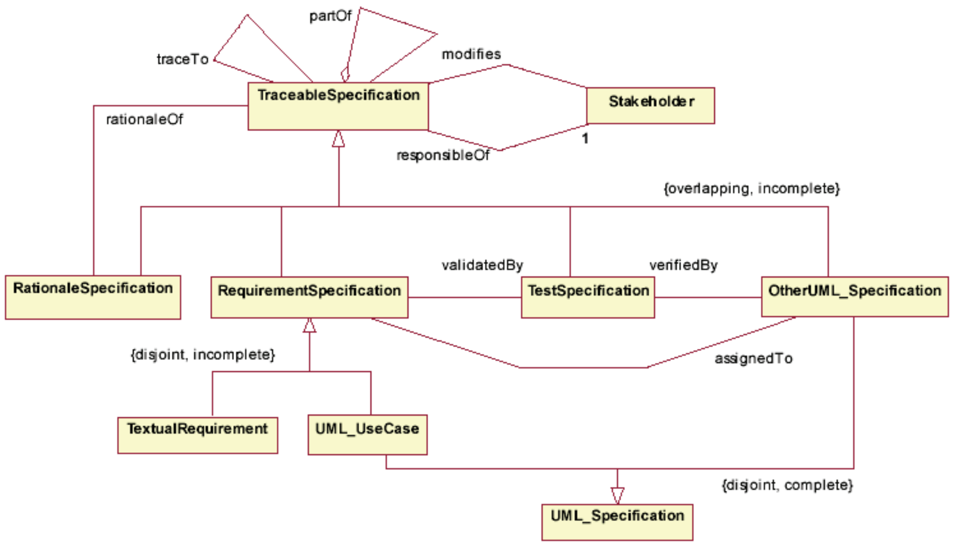


Figure 3.6 Metamodel for requirements traceability (Letelier, 2002)

Two types of entities are concerned in meta-model: *TraceableSpecification* and *Stakeholders*. Stakeholders are responsible of creating and modifying specifications. A *TraceableSpecification* is a software specification with a certain granularity level (e.g., a document, a model, a diagram, a section in a document, a text specifying a non-functional requirement, a use case, a class, an attribute, etc.). The granularity for a *TraceableSpecification* is defined by means of the aggregation with the role name *partOf*.

The type of entity *TraceableSpecification* is a generalization of *RationaleSpecification*, *RequirementSpecification*, *TestSpecification*, and *OtherUML_Specification*. A *RequirementSpecification* is a requirement or group of requirements and, according to how they are expressed, can be classified as *TextualRequirements* or *UML_UseCase*. A *RationaleSpecification* establishes fundaments, alternatives or assumptions associated to a *TraceableSpecification*.

Letelier proposes several types of traceability links. The most generic type of traceability link is represented as *traceTo* which allows establishing traceability links between any *TraceableSpecification*. The rest of types of traceability links (modifies, *responsibleOf*, *rationaleOf*, *validatedBy*, *verifiedBy* and *assignedTo*) are more specific. The link named modifies establishes a relationship between Stakeholders and *TraceableSpecifications* that they modify. In a similar way, *responsibleOf* determines the *Stakeholder* who is responsible of the definition and maintenance of a *TraceableSpecification*. The type of link named *validatedBy* relates *RequimementsSpecifications* with the corresponding *TestSpecifications* that validate them. The type of link *verifiedBy* determines the *TestSpecifications* that verify a UML specification. Finally, the type of link *assignedTo* determines the UML model elements that realize certain requirements.

This approach is based on integrating of textual specifications with standard UML diagrams and integrated to Rational Rose™ using Rational Unified Process™ as a development process. Recall discussion about extensibility of Rational Rose earlier in this chapter.

Ophelia project – Environment for traceability support

The Ophelia project is similar in scope to the settings of this thesis, i.e. both targets to support distributed development and various tools being used. Though they it differs in the method for dependency management.

The aim of Ophelia project (Ophelia, 2003) is about tracing relations among all elements, so that associations can be tracked among any given two objects at any time. This approach is slightly different from other as most approaches deals with tracing associations within requirements and their impact on other project elements.

The goal of Ophelia is to propose a definition of a set of CORBA interfaces for various types of tools used during project's development, starting at requirements elicitation, ending at documentation and test repositories. These interfaces specify abstract functionality of a certain type of tool, but they are not bundled with any particular implementation. Ophelia must have extensive support from tools vendors - they must include an implementation of Ophelia interfaces in their products to allow their coexistence in the integrated platform.

The Ophelia consortium is going to develop a fully functional instance of Ophelia called Orpheus, integrating mostly open-source tools, but also some commercial ones, for which Ophelia plug-ins could be written. Ophelia was designed with distributed environment in mind, Modules (e.g., requirements module interface, design tool interface) composing the platform work in client-server mode. It is not possible to connect an instance of, e.g., ArgoUML, directly to Ophelia. This tool should be connected to a Design Module, which provides all diagrams available in the project, locking of resources (users can not work on the same file at the same time).

All Ophelia interfaces share common definition of an object. Components of a particular instance of Ophelia may obtain information about objects stored anywhere within that instance, regardless of what type of object it is, or which module stores it. Traceability Module (depicted in Figure 3.7) makes use of this feature to store relations among objects in the system.

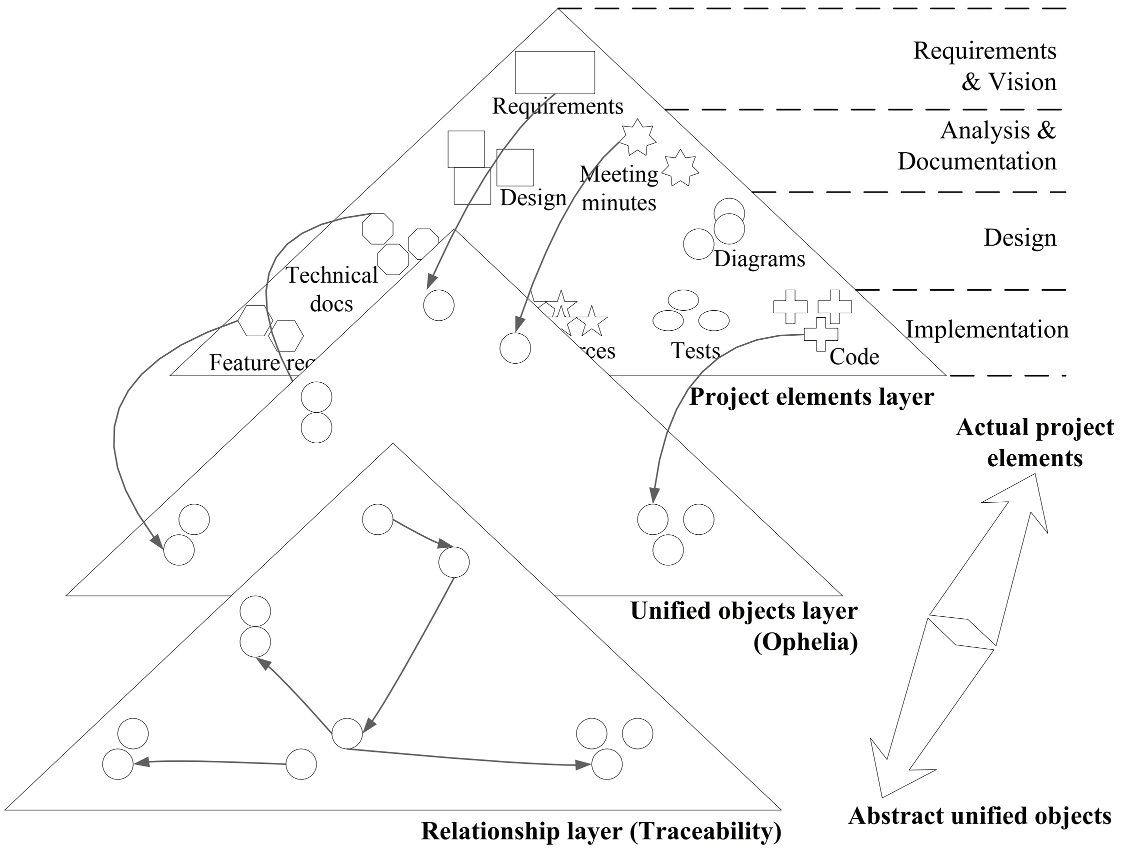


Figure 3.7 The traceability layer of Ophelia (adapted from (Ophelia, 2003))

By now developer should set up relations between new element added to the project and other object already present in the system. Ophelia's integrated traceability approach allows to project members easily understand and measure the size of a particular element's dependency graph and its relation to other parts of the project. Change management is facilitated by combining traceability and messaging - users can be notified about object changes (see Figure 3.8).

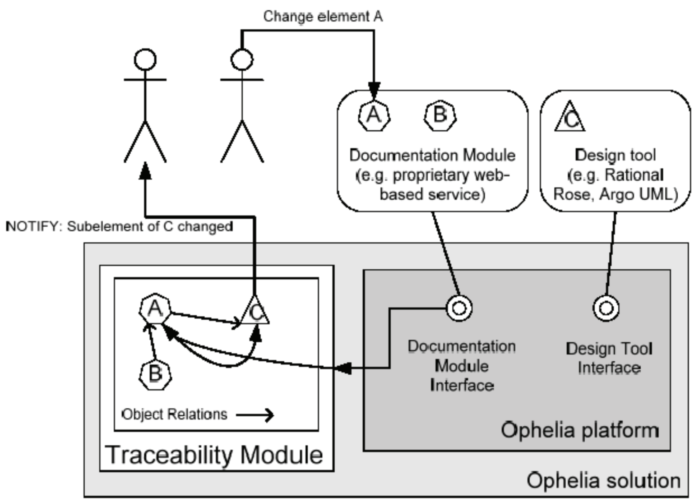

Change element A
A
B
Documentation Module (e.g. proprietary web-based service)
C
Design tool (e.g. Rational Rose, Argo UML)
NOTIFY: Subelement of C changed
A
C
B
Object Relations
Traceability Module
Documentation Module Interface
Design Tool Interface
Ophelia platform
Ophelia solution

Figure 3.8 Propagation of notification in Ophelia approach

Ophelia project intends to contribute by providing open-source repository and environment for distributed software development. Nevertheless with no support from vendors of software development tools the entire idea could fail (all tools used in the project should be integrated in order to enable traceability). Especially as tool vendors would like to have their market advantage over other products - the need of interfaces unification is a contradictory and some trade-off certainly needs to be found.

Scenario driven traceability between requirements and architecture

For the change integration and product evolution by relating architectural descriptions and requirements specifications, (Pohl et al., 2001) proposes:

- To structure trace information by defining orthogonal meta-models which define the concepts and relations about information to be recorded during system development. The meta-models provide the basis for implementing a trace repository;
- To use scenarios as central means for achieving a semantically rich interrelation of requirements and architectural artefacts.

Pohl et al. (2001) define a scenario-centred trace structure which facilitates consistent and effective change integration. This structure consists of six meta-models. Those meta-models are enriched by defining typed dependency links which express the relationships between meta-model components. The resulting traceability structure empowers to capture requirements and architecture information in much more detail and thus support consistent and effective change integration.

This structure is suggested to extend by taking domain and product specific constraints and information into account. Adding domain and product specific concepts and relationships to the generic structure empowers to distinguish domain/product typical features and significantly improve the support for consistent and cost-effective change integration.

Rule-based approach to traceability

Approach by Spanoudakis et al. (2004) is similar to the method proposed in this thesis as both rely on conceptual model (analysis object model as it is called by Spanoudakis et al.) to discover related fragments. However, Spanoudakis et al. use model only for requirements traceability.

Spanoudakis et al. (2004) present a rule-based approach to support the automatic generation of traceability relations between requirement statements and use cases (expressed in structured forms of natural language), and analysis object models for software systems. The generation of such relations is based on traceability rules of two different types – *requirement-to-object-model* rules to trace the requirements and use case specification documents to an analysis object model; and *inter-requirements traceability rules* to trace requirement and use case specification documents to each other.

This approach can generate four types of relations between these artefacts, including the *Overlap*, *Requires_Execution_Of*, *Requires_Feature_In*, and *Can_Partially_Realise* relations. These relations are generated by analysing the contents of the involved artefacts using traceability rules of two different types, namely RTOM and IREQ rules.

An *Overlap* relation hold between:

- sequence of terms in a requirement statement or part of a use case, and a class, attribute, association or association end in the analysis object model, or
- sequence of terms in a requirement statement and a sequence of terms in a part of a use case.

A *Requires_Execution_Of* relation may hold between:

- sequence of terms in a requirement statement, or a part of a use case, and
- operation in an analysis object model.

A *Requires_Feature_In* relation may hold:

- between a part of a use case specification and a requirement statement, or
- between two requirement statements.

A *Can_Partially_Realise* relation may hold between the description, an event (normal, exceptional or triggering) or a post-condition of a use case and the description of a requirement statement. The meaning of the relation is that the execution of the use case can realise part of the requirement statement.

RTOM rules are used to generate traceability relations between textual requirement statement and use case documents and analysis object models. These rules specify ways of matching syntactically related terms in these documents with semantically related elements in the analysis object model. The syntactic relations required by these rules are defined in terms of the grammatical roles of the words in the textual documents which are identified using probabilistic grammatical tagging technique. IREQ rules are used to generate traceability relations between different requirement statement and use case documents or between different parts of the same requirement statement or use case document. Figure 3.9 sketches the approach.

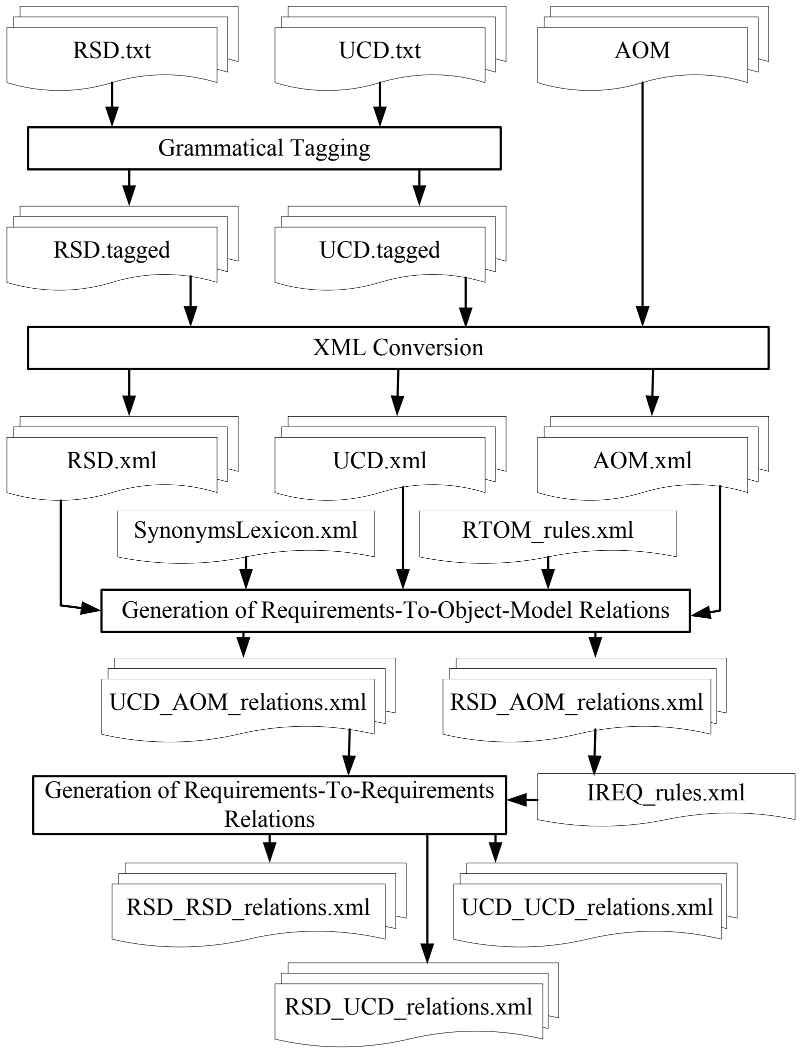


Figure 3.9 The process to establish traceability: rule-based approach (adopted from (Spanoudakis et al., 2004))

Their approach supports the automatic generation of traceability relations using analysis object models (AOM) that specify the main entities in the application domain of a system as well as the parts of it that support the interactions with the users and deliver the expected functionality. Traceability relations are established between:

- requirement statement documents (RSD) which are expressed in structured forms of natural language and define the required functional and non-functional features of a system in broad terms;
- use case documents (UCD) which are expressed in structured forms of natural language and provide a complete and detailed description of the different ways in which a user may deploy the system and specify detailed functional requirements for it.

The traceability relations in the rule-based approach are generated through the process consisting from four stages:

- grammatical tagging of the textual requirement statement and use case documents;
- conversion of the tagged requirement statement and use case documents, and the analysis object model into XML representations;
- generation of traceability relations between the requirement statement and use case documents and the analysis object model; and
- generation of traceability relations between different parts of the requirement statement and use case documents.

The main difference between the IREQ and RTOM rules is as follows. RTOM rules which generate traceability relations based on a direct grammatical analysis of the contents of the involved requirement statement and use case documents. IREQ rules generate relations between these documents only if they are connected with particular combinations of other traceability relations with the same elements of an analysis object model.

Other approaches to traceability

Other approaches can be classified to analytical and post-analytical. The analytical approaches, such as the work by Egyed (2001) and Frezza et al. (1996), use analytical methods after the artefacts are completed to verify that the artefacts fulfil all their requirements. (Egyed, 2001) suggests using a scenario driven approach to acquire runtime information about a system and relate the information – footprints - to the requirements and model of the running system. The footprints are then analyzed in a tool Trace Analyzer, which shows how the components of the system interact when performing specified scenarios. Thus, it is possible to obtain added trace information on how the running system actually fulfils its requirements and which parts of the design are affected. Egyed (2001) proposes to derive traces through (see Figure 3.10):

- **Commonality** - the property of “commonality” allows us to identify trace dependencies among A1 and A2, B1 and B2 by investigating whether or not their footprints overlap. In Figure 3.10 an overlap between A1 and B1 in the footprint {2,3} is present. The tool thus infers the following trace dependencies between A1 and B1;
- **Grouping** - the trace analyzer technique increase the strength of a trace dependency by combining model elements. For instance, model elements A1 and A2 individually only trace to a part of B1 (strength less than 100%), but A1 and A2 together trace to the whole of B1 (see complete overlap of ellipse B1 with the combined ellipses for A1 and A2 in Figure 3.10; strength = 100%). In reverse, B1 still only traces to a subset of A1 and A2 together;
- **Set Theory** – it is possible that not only footprints but also their model elements overlap. If two sets of model elements trace to similar lines of code as in the case above then overlaps in the sets of model elements may also be used to derive more precise trace dependencies (see figure 3.9). For instance, if model element A1 is known to only trace to {1} and model elements A1 and A2 together are known to trace to {1,2} then using set theory it is possible to derive a more refined understanding - it is certain that {2} only traces to A2 and not A1 (set minus) or that {1} must trace to A1 and potentially also to A2 (set intersection). Figure 3.11 shows this relationship graphically.

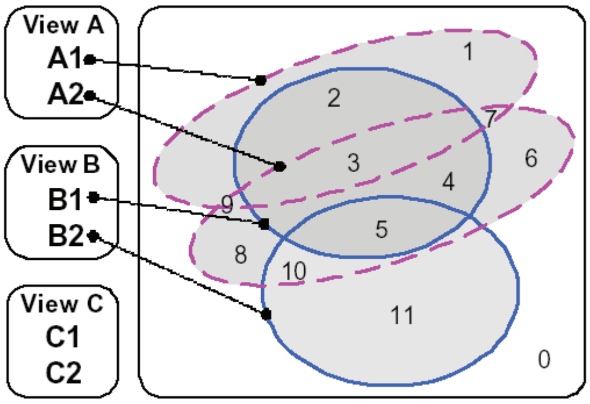


Figure 3.10 Footprints of model elements

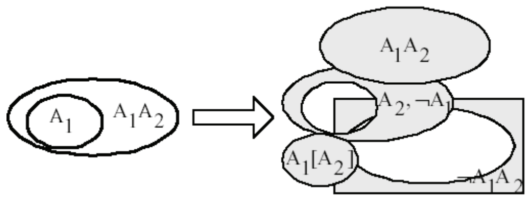


Figure 3.11 Set Theory on trace overlaps

Frezza et al. (1996) on the other hand, propose a system of simulation where both the requirements and implemented system are simulated in order to obtain a set of result data. The data from the requirements and implementation are then compared, which result in a quantitative measure of how accurate the running system implements the requirements.

Those approaches could be characterized as post-analytical, i.e. traces are established after all artefacts are developed and, per se, contribute mainly for product maintenance.

However, there are more framework based approaches (as above described approach by Letelier (2002)) which have been more dominant in the research community. Frameworks, such as the work by Grunbacher et al. (2001) (CBSP (Component, Bus, System, Property) approach) supply the developers with terminology, methods and CASE-tools, which impose a particular structure to both the requirements and the element of other phases, such as design elements, in order to achieve traceability from requirements to artefacts developed.

Grunbacher et al. (2001) propose a framework for refining draft requirements into draft architecture, through a process of selection and assessment. The framework incorporates the notion of differentiating views and supports the recording of rationale. Their proposal deals with refinement of requirements to initial architecture, as requirements may explicitly or implicitly contain information relevant to the system's architecture. The approach is however limited to the transformation from requirements to architecture. Knethen (2002) on the other hand suggests a conceptual trace model and a set of guidelines for using this model. The model separates logical and documentation aspects of requirements structure, in order to obtain a structured composition of requirements and their relations to design elements. The method is primarily aimed at maintenance, where change impact analysis and understanding of existing system is essential.

Furthermore, (Cerbah & Euzenat, 2001) adopt a linguistic view of the requirements tracing, and propose a methodology to allow the tracing of informal text based requirements into formal models through the use of linguistic analysis. They adopt the hyper linkage of documents from different phases of development. If these documents are suitably annotated, they can provide a meaningful design history throughout the development lifecycle and increase the browsing capabilities. This could be done mainly manually or semi-automatically using linguistics technique. Cerbah and Euzenat (2001) have implemented system that generates class hierarchies out of textual requirements specifications and establishes traceability between models and texts through terminology. The authors have not explicitly stated, but the same technique could be use to relate the documentation.

Cleland-Huang et al. (2003) have also developed a system for maintaining dependency relations between requirements and other software artefacts. Their system is based on an event-notification mechanism that implements the observer pattern (Gamma et al., 1995). More specifically, the requirement documents can register their dependencies to other artefacts using the registry of the system. Following the registration of dependencies their system monitors the artefacts and when any of them is modified it notifies all the requirements documents which are dependent on it of the change. The requirement documents have then responsibility for updating their contents if necessary. This system can be used for maintaining dependency relations once they are identified but provides no support for identifying them.

In general, traceability attempts to establish the dependency links and uses them for change management. When it comes to the efficiency, then different authors approach that differently, unfortunately not many of them have evaluated that aspect of their approaches. Overall, requirements *Req6* and *Req12* are satisfied.

3.6 Summary

In this chapter we have conducted the state-of-the-art survey on technological and methodological support for distributed collaborative development. We have taken a look at CASE tools, supporting repository systems including content management systems, briefly discussed CSCW tendencies and relevance. Finally methodological support for managing the variety of development objects has been analysed. Below, Table 3.2 summarises how different technologies do satisfy the requirements identified in chapter 2 as important for cooperative distributed development.

Table 3.2 Summarizing overview of related technology

Requirements	*Main technology*		*Supportive technology*	
	CASE	*CSCW*	*CMS*	*Other techni-ques*[5]
Req1-Flexible access to the product	Medium	High	Medium	Medium
Req2-Unrestricted product fragment types	Medium	High	High	Low
Req3-Unrestricted relation types	High	High	High	Low
Req4-Incremental product fragment refinement	Medium	Medium	Medium	Medium
Req5-Support for boundary objects	High	Medium	Medium	Medium
Req6-Active delivery of information	High	High	High	Medium
Req7-Knowledge externalisation in a means of conceptual domain model	High	Low	Medium	High
Req8-Domain concepts explanation (extension)	High	Medium	Medium	Medium
Req9-Support for knowledge internalization	High	Medium	Medium	High

[5] See discussion in section 3.5.

Requirements	*Main technology*		*Supportive technology*	
	CASE	*CSCW*	*CMS*	*Other techni-ques*[5]
Req10-Conceptual domain model should be available through whole development life-cycle	Low	N/R	Low	High
Req11-Flexible metadata specification about development objects	Low	Low	High	High
Req12-Efficient dependency management	Medium	Low	High	High

Scale used:

High – requirement is fully satisfied;	*Medium* – requirement is partially satisfied;
Low – requirements is not satisfied;	*N/R* – requirement is not relevant.

To sum up, conceptual domain model usually is developed at the beginning in order to sketch down the problem, but is not used in later development phases (with the exception of requirements engineering phase). Only the methods that deal with content annotation use domain model (ontology) throughout the life-cycle of a particular activity. However, this activity is nothing like systems development. There are ongoing research projects (e.g., earlier mentioned (Saeki, 2004)) trying to relate software components to domain specific model/ ontologies.

Overall, there is an interesting tendency of augmenting various open source applications with different advanced functionality. Obviously, the community is realising that it is better to join and add new features to the existing tools (especially supported by big companies), than creating everything from scratch, for instance, Eclipse framework with plenty of different functionality plug-ins, Protégé tool, etc.

A promising research directions is the one started by Sun et al. (1998), augmenting current state-of-the-technology tools by adding collaboration support without a need to change source code of the augmented tool, i.e. CoWord, CoPowerPoint (Sun, 2002). That's promising strategy as users are used to those tools and their functionality. This direction of research (implementation of research results) has advantage against creation of new tools fully supporting collaboration, but otherwise having a limited set of functionality.

4 Repository Objects

"There is nothing permanent but change."
— Heraclitus

This chapter revisits the scope and settings of the thesis, presents an overall method first, then the basic concepts and essential techniques supporting the proposed method are discussed. Finally, the chapter enlightens on underlying repository support. A repository is seen as a mechanism for storing any information about the system specification at any point of a life-cycle. Repository services are meant for extensibility, recovery, integrity, naming standards, and a wide variety of other management functions (Glossary, 2005). Therefore, in distributed development settings repository is a main instrument to store and disseminate information to involved parties.

4.1 Overall Method

Distributed development project consists of several teams (project groups) developing one or more product fragments (Figure 4.1). Each of the product fragments may depend on one or more related product fragments. Some product fragments may be composed from smaller product fragments developed by team members.

The proposed method is illustrated by elaborating each step (*step_O1* to *step_O4*). The overall account of the method is given in Figure 4.2:

Step_O1. Developing conceptual domain specific model. During this step developers produce a model fragments describing their view and understanding of the problem on-hands. Model fragments are stored in a repository. Then concepts mapping is performed and developers are provided with a list of similar concepts, i.e. some initial support is provided to facilitate identification of a common conceptualisation, i.e. view alignment. The developers identify the same concepts and agree about their proper names (alignment of terminology). The proper concept names are stored in a concept space with a list of alias names (synonyms, specified by each of developers). Developers can still maintain the personal view with a preferred terminology. This part of our method is elaborated in chapter 5.

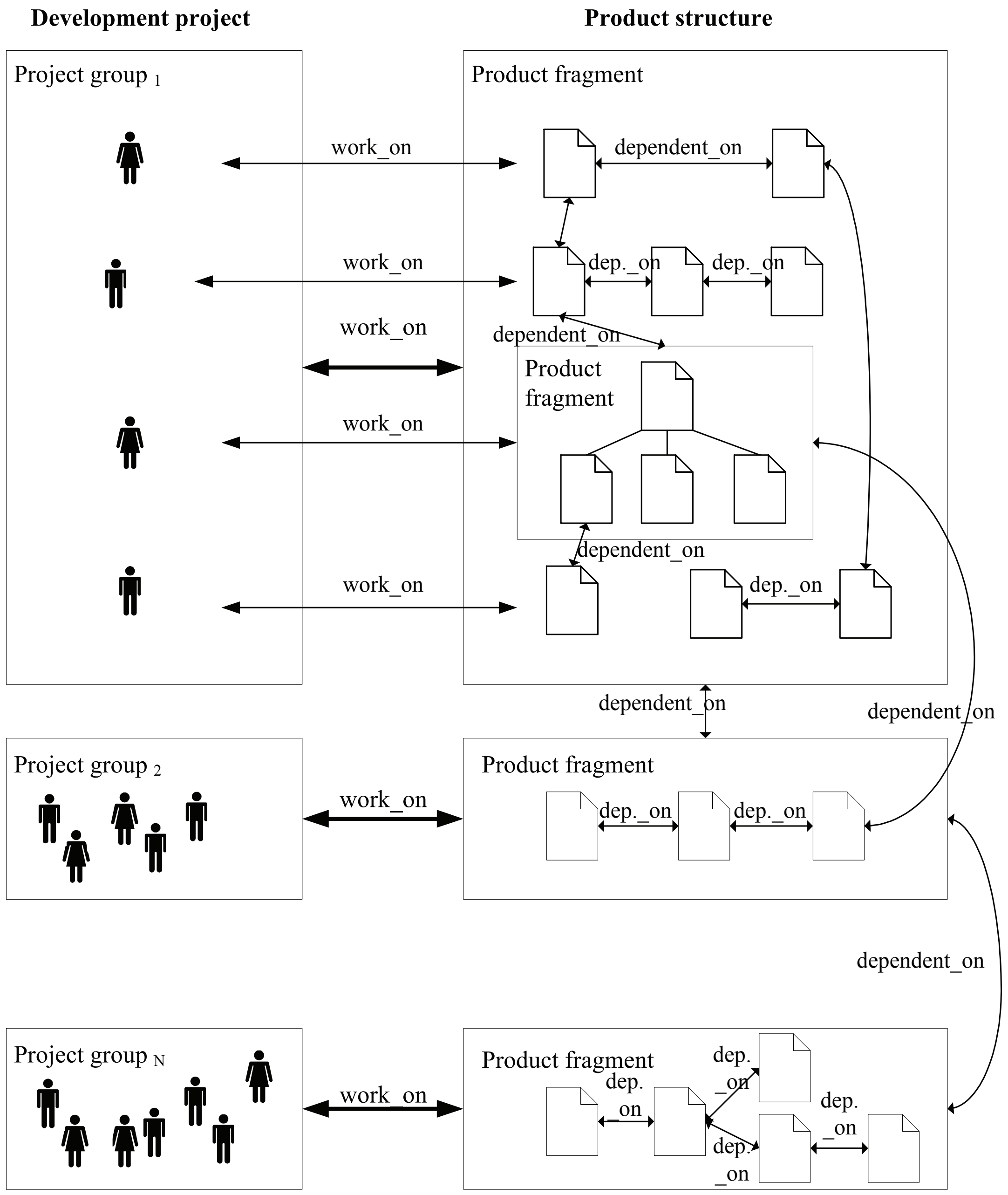


Figure 4.1 Distributed development

Step_O2. Fragments association with concepts. Every developer uploads a product fragment developed by him/her. While uploading it to the repository they relate the produced development objects to the structure of problem they are trying to solve, i.e., by associating produced product fragment with one or more concepts from an earlier defined domain model. An association process here can be treated as a classification of deliverables. Developers can choose confidence level when associating with concepts, e.g., that allows to specify the strength of association. Chapter 6 elaborates on this and the following two steps.

Step_O3. Change impact prediction & fragments management using associations. Thus, dependency relations (relatedness) are based on the semantics of the product fragments. Fragments are associated with the concepts from the domain model. Therefore, all developed fragments are linked through the conceptual domain model as follows. There exists a set of domain concepts $\{C_1, C_2, \ldots, C_n\}$ and a set of product fragments $\{F_1, F_2, \ldots, F_m\}$, then consequently:

- If product fragment F_i is associated to domain concept C_i and product fragment F_j is associated to C_i, then transitively F_i also relates to F_j:

$$(F_i \to C_i) \wedge (F_j \to C_i) \Rightarrow F_i \to F_j \ . \qquad \textit{Eq. (4.1)}$$

- Given, the related domain concepts C_i and C_j, and product fragment F_i associated to concept C_i and product fragment F_j associated to C_j, then dependency to a certain degree exists between F_i and F_j.

$$(C_i \to C_j) \wedge (F_i \to C_i) \wedge (F_j \to C_j) \Rightarrow F_i \to F_j \ . \qquad \textit{Eq. (4.2)}$$

Computation of relatedness degree between product fragments is based on the relationship types within conceptual domain model and association strength (provided confidence level), i.e. "*if two classes have an association between them, then instances of these classes are, or might be, linked.*" (IBM, 1996)

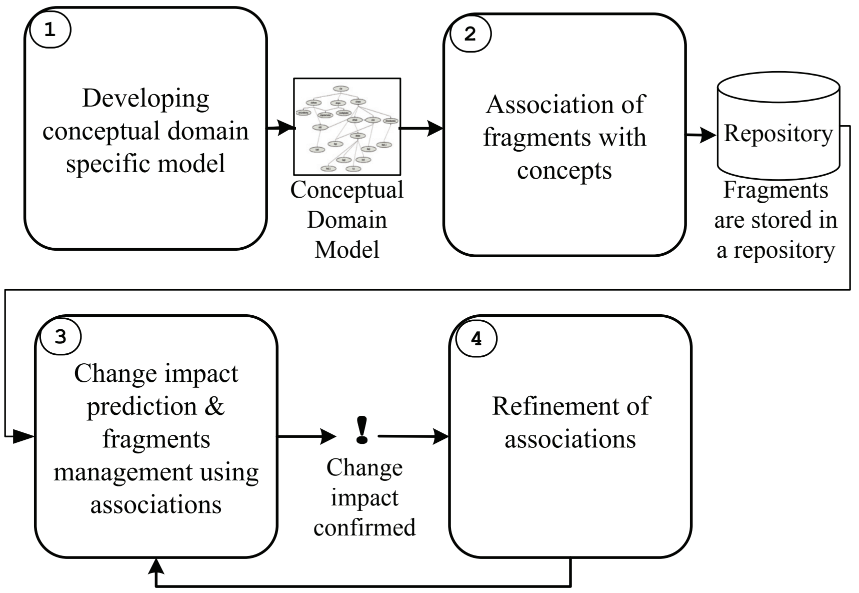

1
Developing conceptual domain specific model
Conceptual Domain Model
2
Association of fragments with concepts
Repository
Fragments are stored in a repository
3
Change impact prediction & fragments management using associations
!
Change impact confirmed
4
Refinement of associations

Figure 4.2 Main functional steps of the method

Step_O4. Refinement of associations. More precise dependency relationship is captured by direct links between related fragments. Since that is not a trivial task even in a small scope projects, we see it important to have them at some certain stage of the project. Therefore, the last step of our method is designated to facilitate direct linking.

Establishment of the direct linking is done in a few steps. The initial one is, of course, the fragment association with a domain concept. Next, exploitation of those relationships is a means for the change impact prediction and assessment (as discussed in *step_O3*). Every change of an associated product fragment produces a list of possibly impacted fragment. Then a responsible developer (typically, a creator of possibly impacted fragment) investigates the change and impact caused. If change impact prediction was proven, i.e., verified, then the developer confirms change impact (dependency) between the two product fragments. The statistics about confirmed and rejected change impact is stored in the repository. After a certain threshold, an establishment of the direct linking between those two fragments is suggested automatically. In this way, we are able incrementally refine and establish direct dependency links between product fragments.

4.2 Repository Support

The repository system simplifies the construction of information systems engineering environments by providing a set of commonly needed facilities, like integration components and support for higher level constructs that are not commonly found in operating systems. Another purpose is to support the porting of environments among different hardware configurations and operating systems.

The repository system provides facilities for incorporation of tools and thus provide generic utilities that improve integration. Incorporation in this context has three aspects: interoperability within life-cycle phases, interoperability across life-cycle phases, and interoperability across a distributed development environment.

Interoperability within life-cycle phases focus on the various models used during one specific life-cycle phase in order to enforce consistency within each phase. For instance, integrating DFD and ER models during the analysis phase will ensure consistency between flows and datastores in the DFD and entities in the ER model. This aspect of integration will increase productivity and quality of the product developed in the particular phase (see *Phase product* in Figure 4.3) of development life-cycle.

Interoperability across life-cycle phases focus on the specifications created by the different tools that used in different phases of the development life-cycle. It is desirable that output from one tool can be automatically supplied to another tool. This may be realised provided that the concepts manipulated by the tools can be related to each other. This may decrease the need for manual intervention and eliminate a possible source of errors, to increase productivity and quality in the development life cycle. Though it is important, it is difficult to do, as random tools might be used in such type of projects.

Interoperability across a distributed development project improves communication and coordination among stakeholders of the project. Different developers are working on related sets of specifications. They need to synchronise their work. When the specification sets overlap, the need for communication and coordination increases.

Those three aspects defines the special need for accommodating huge variety of different development objects and interrelating them at least at data storage level, if not providing seamless transition from one to another tool.

4.2.1 Repository functions

During the development, many development objects of many different types are defined, created, manipulated, and managed by a variety of tools that need to be shared. In that context, main repository functions are as follows.

Version and configuration management. Product fragments are constantly updated and the repository needs to store snapshots of that product fragments at different times. Configurations allow the developer to group related versions into sets that have a common purpose. Together, version and configuration management support team development by helping developers to manage cooperative activity.

Relationship management. By establishing different types of relationships between development objects, developers can locate related sets of objects, and can track dependencies in the deployed components.

Schema management. By providing facilities to create and modify object types.

Query. By querying the contents users can browse the repository.

Special DBMS requirements for CASE. In addition to the features usually supported by a DBMS as non-redundancy of data, data independence, queries, real-time updating, locking, concurrency, integrity, etc., the CASE application require special attention to the following areas: (1) handling multiple versions of specifications, (2) maintaining dependencies among development objects created by different tools, (3) enforcing integrity constraints to ensure that the database remains consistent and meaningful, (4) providing flexible access of the development objects.

4.2.2 Repository object types

Model in Figure 4.3 defines a repository and objects types to be stored in it. *Product* is developed using *system development tools* (*Syst.Dev.Tool*), where a *system development tool* can also be seen as *product*, when it is under development. Every product development has a specific *lifecycle* consisting of different *phase type* (e.g., business analysis, requirements engineering, design, implementation, testing, etc). Each *phase type* has a distinct *phase product* (e.g. requirements specification, design, code, user manual, and software itself), which is result of particular *lifecycle phase*. A *product* is final result of the development *project*, and it consists of the interrelated *phase product*.

Product fragment is a semantic piece of phase product. An information system is viewed as a product composed of product fragments, which are of the following types:

- *Model fragments* – sub-models of a conceptual model of the information system being under development. Only the semantic content of the model is stored, not diagram layout information;
- *Diagrams* – stores layout information of the conceptual model view. Diagrams may exist in several different layout versions without affecting the conceptual content;
- *Code fragments* – code modules (files);
- *Document fragments* – pieces of the documentation of the models and code fragments.

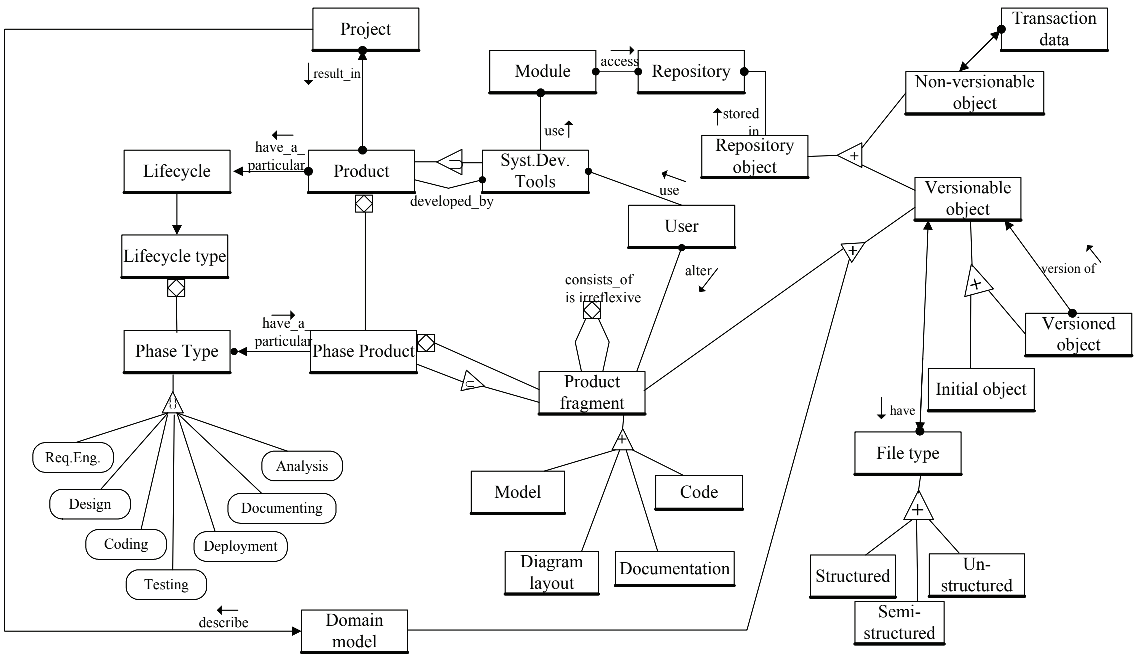
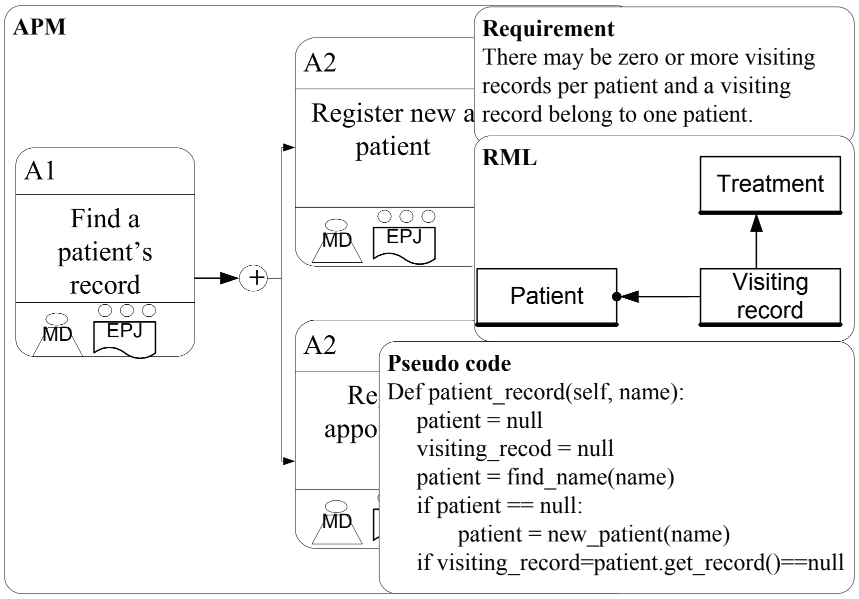


Figure 4.3 Main concepts of ISE

All these product fragments are stored in files either as structured (e.g., model fragments and diagrams), semi-structured (e.g., code and document fragments) and unstructured (binary, e.g., figures) information. Figure 4.4 illustrates different types of product fragments. A model structure depends on a modelling languages used, diagram structure reflects the basics of visual languages, code structure is expressed in programming language and document structure reflects common document architectures.



```
Def patient_record(self, name):
    patient = null
    visiting_recod = null
    patient = find_name(name)
    if patient == null:
        patient = new_patient(name)
    if visiting_record=patient.get_record()==null
```

Figure 4.4 Different types of product fragments

Repository objects are either versionable or non-versionable. *Non-versionable* repository objects are transactions data (the data is generated automatically during the manipulation of the repository content). *Versionable objects* have file structure type being structured (graph, e.g. model in XML), semi-structured (has no clearly defined structure, but it is possible to reason about part of the structure, e.g. text files and paragraphs) or unstructured (binary). Versionable object in the repository is an *initial object* (i.e. an initial version of the versionable object), and a version of the initial object or previous version. Product fragment can be composed of smaller product fragments. Product fragment is a versionable object.

Domain model is specific product fragment as it describes the domain of the project and is used to inter-relate all product fragments by associating versionable object with concept(s) from domain model to specify semantics of the fragment.

The development tool accesses the repository through module to manipulate the repository objects stored in the repository. Module acts as middleware between the development tools and repository.

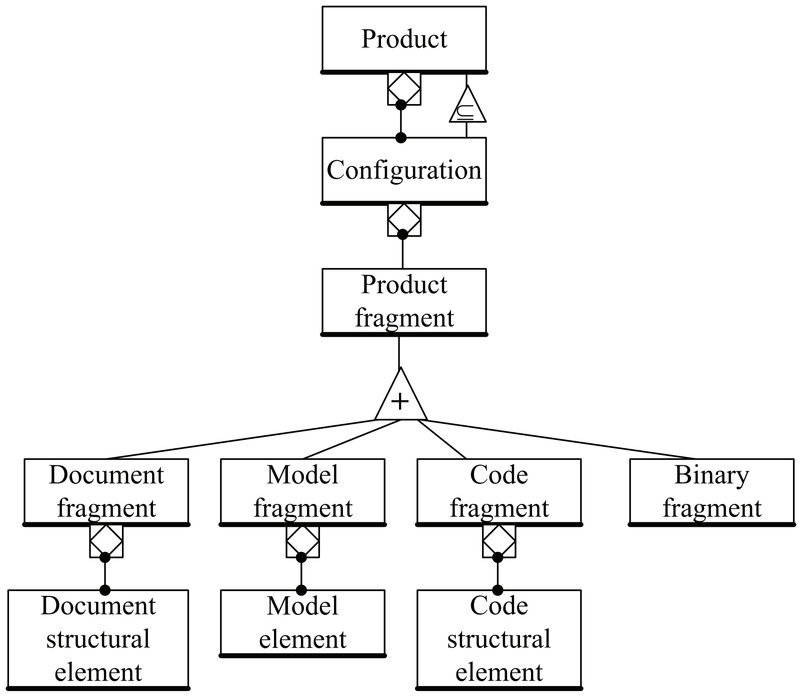


Figure 4.5 Granularity and structure of product fragment storage

Figure 4.5 illustrates product fragments composition and storage structure in the repository. Smallest unit in the repository is a model element, whereas the biggest “container” is project, i.e. modelling is done within the scope of the project. Meaning, that the project must have configuration and configuration should be composed for the project. Configuration must contain a product fragment, which could be empty. Product fragment itself is composed of corresponding elements, e.g., document is composed of chapters, sections, paragraphs; code fragment is composed from classes, functions.

Development objects are identified to be of a certain datatype. The repository uses this information to store objects in an appropriate structure and format. The repository maintains datatype information about stored objects, and uses this information in order to convert between storage representation and application-level data formats. For instance, only conceptual part of model fragment is stored in tables, while layout changes are versioned, but stored in complete file on file system.

4.2.3 Information about object

Figure 4.6 defines “enriching” *versionable objects* (development objects) stored in the repository. *Versionable object*:

- has associated *metadata*, describing its properties, i.e. subset of Dublin Core;
- is associated to the *concept(s)* from *domain model* describing semantics of the *versionable object.* Where *concept* is a constituent part of *domain model*, it is used to relate all *versionable objects* and in this way organize and manage them;

- is related to other *versionable objects* based on dependency between them (through *Direct_relationship*);
- is described by attributes such as *ObjectID*, *Description*, *Type*, *CreateTime*;
- has associated *trace info* such as *ID*, *Rationale*, *Change description*, for process traceability;
- might be discussed in a *forum* item attached to it;
- is related to earlier and following revisions.

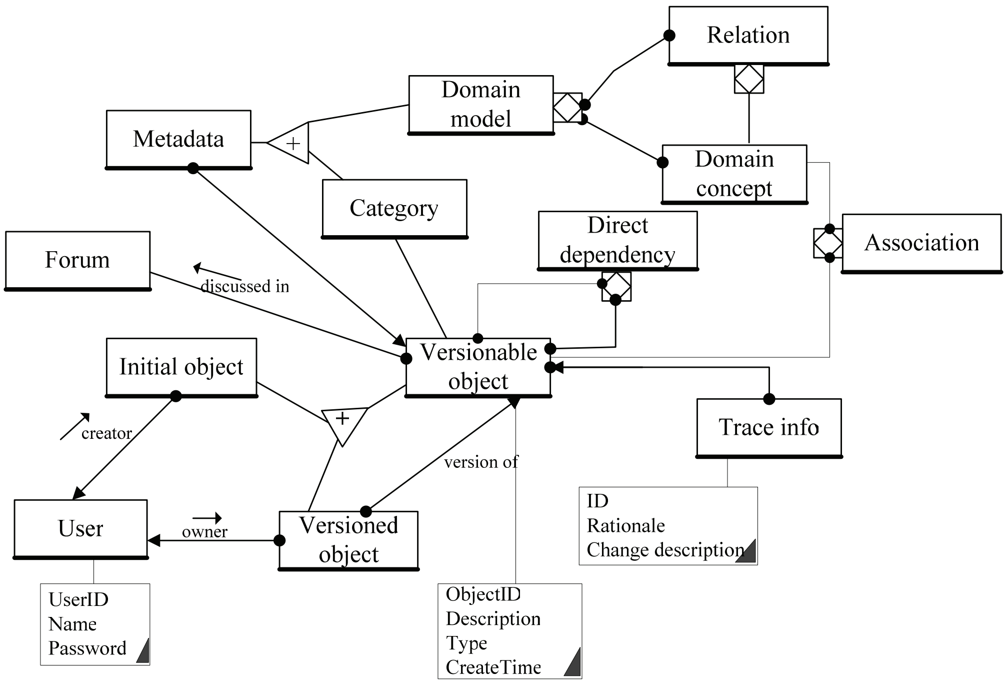


Figure 4.6 Rich information about development object in repository

Both categorisation according to life-cycle phase development object belongs to and *domain model* are treated as *metadata*, explicating the meaning of the development object. *User* is a creator of an *initial object* and owner of *versioned object*.

All these information is modelled as follows. Let *I* be a set of the internal identifiers, *N* be a set of the names, and *V* be a set of the values, e.g. numbers, strings, blobs, etc. Objects are modelled as triples defined below, where $i \in I$, $n \in N$, and $v \in V$:

- Atomic objects *as* <i, n, v>, e.g. <i_5, referent, “Versioned object”>.
- Link objects *as* <i_1, n, i_2> that model relationships between development objects.
- Complex development objects (composition of development objects) *as* <i, n, S>, where S denotes a set of development objects, e.g., <i_9, modelFragment, {<i_3, referent, “A”>,<i_4, referent, “B”>,<i_3, justAlink, i_4>}>

4.3 Namespace and Object Identity

Object identification is a crucial issue in computer science. Inappropriate use of naming schemes can cause serious flaws in a repository design. To manage such a diversity of development objects, descriptive information about the development objects, and composition of the objects a robust naming schema is necessary. A namespace is a set of names complying with a given naming convention. The operations allowed on development object names are performed in the context of a naming mechanism. Development object names are used to refer to objects; to provide information about those objects; to locate development objects given only their names; and to access those objects.

Bunge's principle of nominal invariance (Bunge, 1977) states: "A thing, if named, shall keep its name throughout its history as long as latter does not include changes in natural kind – changes which call for changes of name."

4.3.1 Requirements for object identification

In other words, Wieringa and de Jonge (1995) identify the requirements which should be satisfied by an object identification system:

- **Singular reference.** A naming scheme N satisfies the singular reference requirement if in each possible state t of the world, each proper name in $dom(N_t)$ refers to exactly one object in O (a set of all possible objects);
- **Singular naming.** A naming scheme N satisfies the singular naming requirement if in each possible state t of the world, each object in $range(N_t)$ is named by exactly one proper name from V (value space).

Where domain and range of a naming relation $N_t \subseteq V \times O$ are defined as follows.

$$dom(N_t) = \{v \mid \exists o \in O : \langle v,o \rangle \in N_t\},$$
$$range(N_t) = \{o \mid \exists v \in V : \langle v,o \rangle \in N_t\}.$$

In order to represent historical information adequately, additional two requirements are imposed (Wieringa & de Jonge, 1995) as follows.

- **Rigid reference.** After each state transition of the world, each proper name remains referring to at least the same object(s) as before.
- **Rigid naming.** After each state transition of the world, each object remains named by at least the same proper name(s) as before.

Additional requirements for object naming in the context of objects composition are brought up by (Ramazani et al., 1998). They are as follows.

- Should be possible to check whether two objects belong to the same composition;
- Naming should reflect sharing of objects between compositions;
- Should be possible to reference from one composition to another.

Mainly there are two alternative ways to obtain global name uniqueness:

- a flat namespace where the name uniquely identifies the development object no matter where it is being used.
- a hierarchical namespace where names of development object are qualified with the names of hierarchical superior development objects.

The simplest solution is a flat namespace. Though, hierarchical namespace would allow bigger number of names for development objects and makes it easier to manage names. For instance, development object identification used by Andersen (1994) is encoded as follows.

$$\mathrm{A}\#.\#\{\tau\} < \alpha > .\lambda(L)$$

Where A is abbreviated id for object, #.# denotes revision, τ identifies the transaction, α identifies different abstraction, λ identifies revision number local to the transaction, and L identifies change of layout.

Section 4.4.1 introduces our naming scheme to unique identification of development object.

4.3.2 Co-reference and management of sameness

Here we will focus on the problem of co-reference. Co-reference is the problem that arises when two or more names refer to the same thing/person: *IJHCS* and *Int.J.Hum.Comp.Studs*; *N.A.M.Maiden*, *N. Maiden* and *Neil Maiden*.

Given a set of object, some of them might be the same. For instance, having several photos we can find out that a person on them is the same. We can say that person in photo A1 is the same as person in photo A2, then somebody else may notice that person in photo A3 is the same as person in A2. It is easy to establish the “sameness” links between them (e.g., see left part of Figure 4.7). But if later we discover that A3 is not the same as A2, what should we do with the link between A1 and A3. To facilitate management of co-reference, a new object is created and equality relations are established between real objects and the reference object. A common (proper) name (e.g., ‘A’) can be used, though that is not mandatory. The reason for having alias names is discussed within the context of model fragment management in chapter 5.

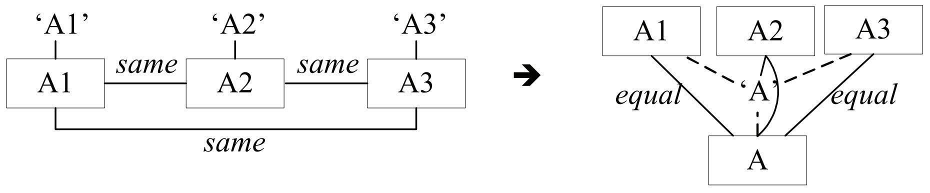


Figure 4.7 Management of “sameness”

4.4 Versioning Framework

Version and configuration management keeps track of product fragments' versions developed by several developers working in a geographically distributed development environment. A version of a product is an immutable, identifiable edition of a product. A product is composed of a number of product fragments. A product fragment may either be a hierarchical composition of other (sub-) fragments or a flat structure with no hierarchical relationships among the (sub-) fragments. A version of a product is a composition of versions of product fragments. We distinguish between 4 versioning dimensions:

- historical versioning, i.e. revisions;
- logical versioning, i.e. variants/branches (alternate, substitute, option);
- view versioning, e.g., informal, semi-formal, formal representations of the same model fragment; as well abstracted or filtered views;
- layout versioning - re-location of model elements, in model fragment case. A new layout version does not change the meaning of the model fragment, just redraws it in a different (more comprehensible) way.

4.4.1 Version identification

Each developer should have direct access to all needed objects. But changed version should be kept with access forbidden for other developers during modification, because the state of fragment is inconsistent in a modification phase. Recall situation discussed in chapter 2 where *n* developers change the same object concurrently, this object should have *n+1* different copies (Estublier, 2001). It means that each developer needs the private copies of fragments.

Versions are usually identified by identifier consisting of the object name and a revision id. Applying this system to development objects yields **dev_obj.0**, followed by **dev_obj.1**, **dev_obj.2**, etc. This way of identification functions well until variants are introduced. The system degenerates quite quickly when variants are common (Fidjestol, 2005). Usual way of handling variants is to modify the revision id to include more information, for instance, inclusion of branch identifier and a new generation count yields **dev_obj.2.3.6**. This revision id identifies sixth revision of the third variant of revision two of a particular development object. Revision id will be extended by two new numbers if a new variant of this development object will be needed. It has been argued by Fidjestol (2005) that this kind of identification system is not practical in a "variant-friendly" environment and in a system where a finite length of the revision id is preferable, for instance, a system implemented in a RDBMS.

Using a direct "version of" relationship instead of a generation count will help to solve the above described problem. From project management perspective the product is treated as consisting of families of development object (product fragments) versions. The content of the object is not interested from the project management point of view. It is important to manage versions. Usually project consists of its members, its development objects and its configuration that is understood as a grouping of object versions for various development processes.

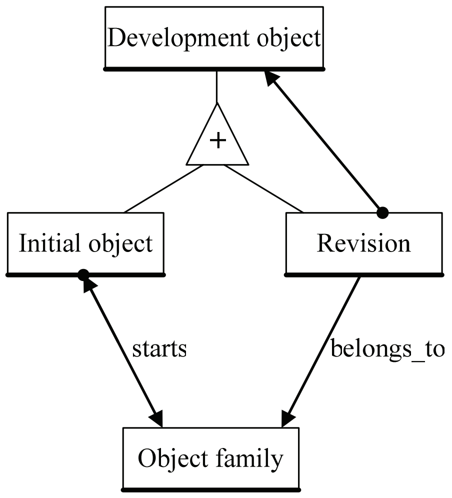


Figure 4.8 Object identity

Each development object may exist in several versions. All versions of an object present different attempts from different developers at different times to design a specific development object or fragment. As all versions represent evolutionary snapshots of the same basic development object, all versions are related. The relationship between two versions is either a revision or a variant. A revision reflects a development history where the most recent version replaces older one. A variant reflects development history where two versions co-exist one replacing other. A collection of all related versions of the object is called family (Carlsen, 1997). In order to place a development object in a version graph, we describe every development object using a triple <*object, family, parent*>, see Figure 4.8 and Figure 4.9.

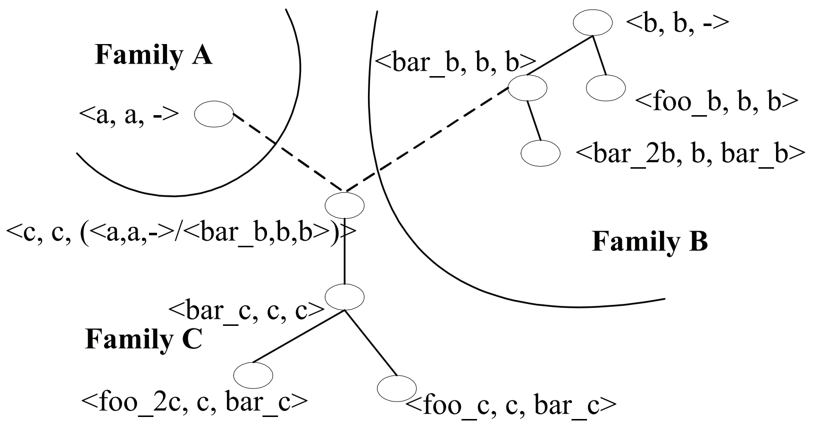


Figure 4.9 Object family and merge of objects

Since new revisions are constantly introduced, to manage variety of information denoted in Figure 4.6, a flexible linking mechanism is necessary, in order to connect new versions. Identifying objects (model elements) by unique family id, i.e. revising an *object A* (*v0.0*) to next version (*v0.1*) does not change it's family belonging, i.e. family id is kept the same. Consequently, describing relation between objects A and B, i.e. $R_{AB} \subseteq A \times B$, using object family id will preserve the relation between particular versions of objects *A v0.1* and *B v0.0*

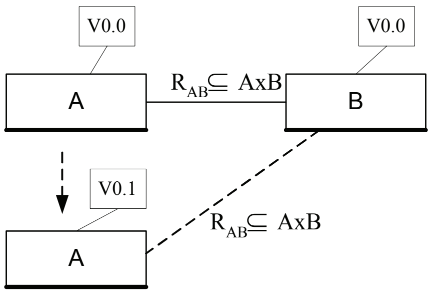


Figure 4.10 Family id and relationship connecting objects

4.4.2 Composition identification

We extend naming scheme by a composition id, actually family id of complex object. For instance, Figure 4.11 illustrates possible three-level composition, where object A is composed from object B, B from C and C has atomic object (element) D.

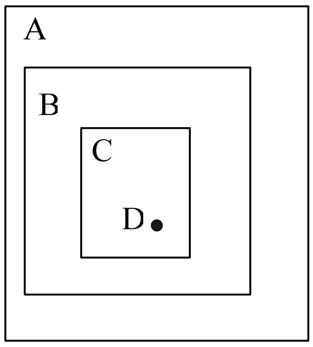


Figure 4.11 Object composition

By extending earlier mentioned object identity triple to quadruple: <*object, family, parent, Composition*>. Composition here is a set of family ids of complex objects (compositions) where a particular object is included. The composition in Figure 4.11 is described as follows.

There is an initial version of object A <a, a, -, {}>. Object B is included in the object A <b, b, -, {a}>. Object B is composed from object C <c, c, -, {b}> that is composed from object D <d, d, -, {c}>.

In this way we are able to propagate change to compositional object, after a component has been changed, i.e. new version of object D will automatically create new versions of C, B, and A. This is discussed in chapter 5 in context of model fragment reuse in (import to) other model fragments.

Every product fragment (development object) is under control in our versioning framework, in addition, structured product fragments have fine-grained versioning control, i.e. all leaf-nodes in Figure 4.5 are under control of the versioning framework.

4.4.3 Configuration identification

Authorised and temporary versions are distinguished. Authorized versions are agreed baseline versions for further work. Temporary versions are created as needed by the developers. When new authorised version appears, all temporal versions could be discarded, or remain saved for the record.

During system development the project is reflected as a set of family trees. Some development object versions of a family may be authorized, others not. Some object versions may be consistent with others. Some object versions represent the latest changes made. A developer must have the possibility to select desired object versions from a family. Organization of development objects in configurations make possible a consistency check with newly created object relative to an existing set of consistent objects and provides the latest set of authorized versions available.

In (Henriksen et al., 1997) it is stated five different properties of configurations must be maintained during systems development:

Authorised. A configuration is authorised if and only if all of its objects are defined as being authorized. An authorized configuration is also called a baseline.

Consistent. The objects of a consistent configuration have to be consistent, both relative to each other and as individual components. A configuration remains invalid if it is not checked or if the consistency checker detects inconsistencies.

Latest. Each object of a latest configuration must be the latest version of its own family.

Owner. The property owner constrains the configuration to contain only object versions owned by a given user. Authorized versions are considered as owned by all project group members.

Project-wide. A project-wide configuration must include one object version from every family within project.

Three configurations are important: 1) the *latest project-wide*; 2) *latest consistent*; 3) *latest authorised*. The latest project-wide baseline reflects all new developments in the project. Latest baselines are not supposed to be consistent, but all new ideas since last logon are detected by inspecting this configuration. Latest consistent baselines represent the most recent stable work. These configurations are the candidates for authorisation. There may exist several latest, authorised configurations, but usually exists one. This is the last configuration, which the group agrees on, and it forms the official basis of all subsequent work. Traceability technique between different versions of the product fragments with incorporated configuration management facilitates extraction of relevant configuration.

4.5 Summary

First we have presented an overall method and main concepts. Then we have discussed variety of object types produced during the development life-cycle and different additional information needed to better communicate the content of various development objects. As a change is constant in such scale projects, have introduced naming scheme for more flexible naming of versions, as well as referencing object composition.

To summarise, the repository is vital in order to support systems engineering in the areas as follows. *Team development* to facilitate developers to manage concurrent activity on different versions and configurations of IS development. *Reuse* to facilitate sorting, storing and locating relevant product fragments. *Dependency tracking* to facilitate establishing and querying relationships between product fragments. *Tool interoperability* to facilitate developers to move easily between tools across the development life-cycle and to manage related product fragments. *Product fragments management* to provide metadata for a project and a library of product fragments.

5 Model Fragment Management

"Many different views of the world may co-exist, each view serving different purpose and/or different people. No view is more correct than another because each view serves a worthy purpose"

— Arne Solvberg, 1999.

As discussed in the previous chapter, our method is centred on the collaborative effort on defining the domain model. The true collaborative aspect of modelling is to enable discussions and awareness of issues and mismatches in the model fragment among the modellers working on it. It is important to have a common vocabulary as well as common understanding of a problem domain.

This chapter elucidates on a part of the method for model fragment management in collaborative settings. Where teams of modellers work in parallel on different parts of a common product model. The work is typically logically and/or geographically distributed. The modellers contribute to the shared domain model, either by posting revisions to previous models, or by creating new model fragments. Either way, others' work is impacted, as externalised knowledge changes social reality.

The chapter is further structured as follows. First, we take a closer look at the modelling process and define a framework for collaborative modelling. Then we go through basic steps of conceptual modelling, i.e. externalisation, internalisation, and commitment. Later, we present model configuration management since the collaborative modelling support is centred on composition of the work, instead of coordination as in the cooperative work. Before summarising the chapter we list a set of requirements for model fragment management.

5.1 Framework for Collaborative Distributed Modelling

In a distributed process, the variability of the model versions increases due to the highly interactive and iterative nature of the development process and to the different, sometimes conflicting, angles to a problem and solution taken by the different stakeholders. Therefore, modelling process can be viewed as three dimensions of requirements engineering (Pohl, 1993): agreement, representation and specification dimension. The *agreement dimension* concerns reaching a common view by beginning from a personal view; domain description moves along the *representation dimension* typically from informal natural language descriptions to more formal representations; the *specification dimension* is traversed from opaque and partial views towards comprehensible and complete view (model) of a problem.

Because of diversity of stakeholders involved, especially socio-cultural distance between them (Agerfalk et al., 2005), the representation dimension is very important in distributed collaborative modelling. Providing a support for seamless transition from informal natural language descriptions of domain to semi-formal graphical models and then, even further, to formal models would facilitate an engagement to modelling process. Stakeholders without previous modelling experience would most benefit from such support. Though it is an important feature for modelling framework, we will not investigate its support in this thesis. Here we focus on externalisation of knowledge and commitment to the explicitly expressed knowledge (in a form of conceptual model). After a model fragment has been created, it is eventually shared among the stakeholders, and then the shared model fragment transits through three major states of knowledge sharing identified by Hoppenbrouwers et al. (2005b). They are as follows (Hoppenbrouwers et al., 2005b).

Aware – a stakeholder is aware of knowledge (model) shared by other stakeholders. Then the shared knowledge (model) is internalised;

Agreed – a stakeholder decides whether agree or not to the shared knowledge;

Committed – a stakeholder decides to adopt her future behaviour according to a particular knowledge (model).

The ultimate goal in distributed modelling is to arrive at a coherent, complete and consistent description of the problem domain. Furthermore, all stakeholders should commit to the common definition. Figure 5.1 illustrates a framework for collaborative distributed modelling (inspired by (Pohl, 1993) and (Hoppenbrouwers et al., 2005b)), and its three axes[6] which are further elaborated in this chapter.

The collaborative modelling framework is fulfilled with a modelling activity that is further elaborated into more detailed iterative steps, as follows.

[6] As discussed in this section, a representation dimension (axis) is important, but out of the scope of this thesis. Therefore, the representation axis is excluded from the proposed framework.

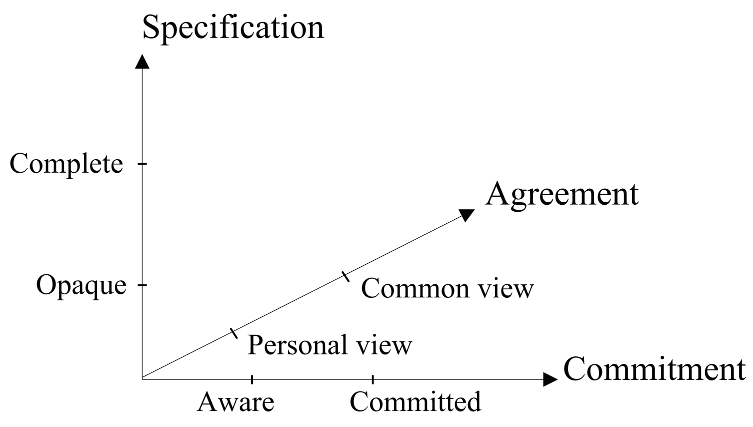


Figure 5.1 Framework for collaborative modelling

Step_M1. Externalisation. After a problem has been identified, it needs to be described and modelled. This step concerns creation of the model fragments, representing stakeholder's view and interpretation of the problem space, i.e., externalisation of own knowledge.

Step_M2. Internalisation. Here different views are compared, aligned and validated against knowledge of other stakeholders. Discrepancies between different views (model fragments) are negotiated and clarified.

Step_M3. Commitment. After agreement has been reached among the stakeholders involved (i.e., amount of discrepancies between individual and organisational realities has been minimized), then stakeholders explicitly commit to the domain model.

The three dimensions of the framework are interwoven with the modelling steps described above. Whole modelling cycle might be iterated several times for the stakeholders to be satisfied. When knowledge of the modeller is first externalised (specified in a model fragment) it usually presents an opaque and personal viewpoint. The model fragment may be internalised only if other project members are aware of its existence, i.e. the model fragment should be made available for others in a common information space (a repository, in our case). Then the model fragment is internalised, i.e. investigated and compared to own perception by other stakeholders. Internalisation results in either agreement or disagreement, both may be partial, i.e. stakeholders may find relevant only a part of the model fragment. Either way, other stakeholders externalise their new knowledge body by developing a new model fragment or refining the previous model fragment.

That continues till complete, common understanding of the problem is achieved. Naturally, stakeholders should commit to the collaboratively developed model. This assumption is restrictive, as a common understanding (absolute agreement) is hardly achievable in distributed settings, especially among socio-culturally different stakeholders. Nevertheless, agreement can be achieved, if not in a form of absolute, then pragmatic commitment.

For instance, in managerial (business and organization management) science the importance of explaining and communicating the overall objectives of organization to its members is vastly discussed (cf., Daft (1995) and Hatch (1997)). Despite of that issue being addressed by top management, there are still employees whom do not commit to the organisational objectives. Anyway, they are able to work for the good of organisation (Poole & Warner, 2000), i.e. pragmatically driven. Similarly, in a development project pragmatic agreement and commitment are feasible and easily enforceable at a certain time point.

5.2 Externalisation

Externalisation as part of the specification dimension deals with the degree of problem understanding. The dimension has the goal to improve an opaque problem comprehension into a complete specification in a form of a conceptual domain model. Every stakeholder externalises own comprehension in a model fragment describing a particular concern within a problem space, i.e. fragmentation of the domain in question. Main activities in this dimension are as follows.

5.2.1 Concept specification

Concepts (among other things) are in general language independent (words "bicycle" and "dviratis"[7] denote the same concept). Concepts are mental or logical representations of reality; they are related to other concepts. Usually, concepts hold symbols but hold them for means of communication. Concepts have intensions and extensions, for instance, "Evening star" and "Morning star" that have different meanings (intensions) yet both refer to planet Venus (extension).

Relevant domain concepts are specified during this stage. They are defined in terms of their (cf., Bleeker et al. (2004)):

- Meaning, e.g., bicycle is a wheeled vehicle that has two wheels and is moved by foot pedals;
- Relationships to other concepts:
  - Compositional definition (Figure 5.2);
  - Taxonomical (hierarchical relational) definition (Figure 5.3);
  - Other relationships with varying semantics;
- Possible names used to refer to them, e.g., {bicycle, bike, dviratis}.

Concepts should have specified a domain specific human readable definition. The purpose of such a definition is to provide explanation of the concept in a natural language to other stakeholders. Goal is to utter as possible clear meaning of concepts, achieve shared understanding of the concepts meaning, and have a

[7] Lithuanian word for "bicycle".

set of possible terms. A concept can have a number of textual expressions, which may differ in their grammatical construction, terminology, and language. Definition of synonyms here is an option, not obligation. We see it important to allow users to use their own vocabulary and not be bound to some "standard" vocabulary. Maintenance of proper concepts names and their synset as common and private views on the model fragment are discussed later.

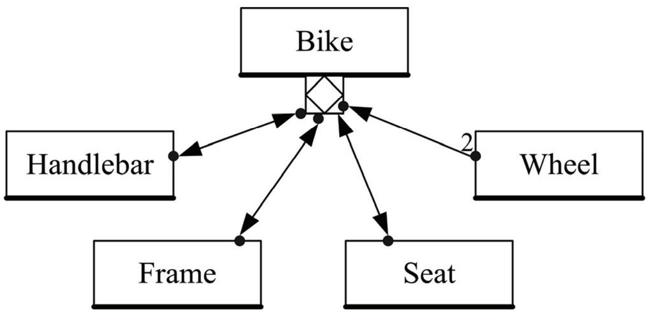


Figure 5.2 Compositional definition of concept "Bike"

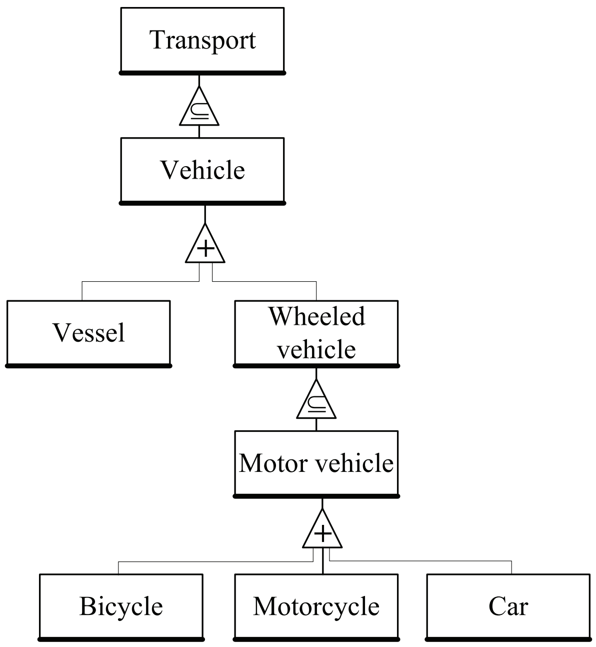


Figure 5.3 Taxonomical definition of concept "Bicycle"

The purpose of concept specification here is to define a scope of domain using terms which are acceptable for stakeholders, a kind of unified vocabulary for stakeholders involved in a particular project. We target to create a lexicon for both relational and non-relational concept, as in Figure 5.4.

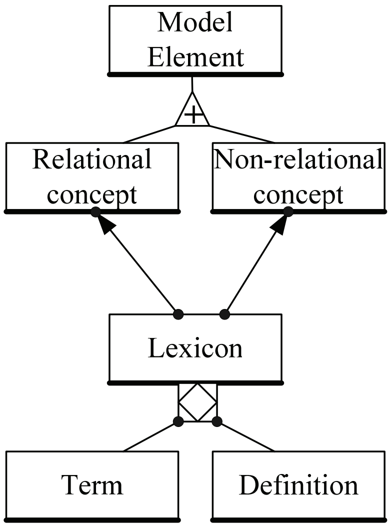


Figure 5.4 Model element lexicon

5.2.2 Model fragment scoping

Purpose of a model fragment is to define a particular phenomena or a limited set of phenomena in UoD. Similarly, a database view provides exactly that: users specify a query that extracts a portion of database instances satisfying the query, creating a specific view on the data in the database. Therefore, we call a part of a model a *model view* or *model fragment*.

Figure 5.5 defines *model* being a set of the *model fragments* which is composed from one or more *model fragments*. *Model fragment* is a *statement* about a part of UoD and is composed of *model elements* which are *modelling constructs* from particular *modelling language* defined by *metamodel*. *Model* complies with its language. *Model fragment* may be composed from other *model fragments*. *Filtered views* are generated from model fragments to enhance comprehensibility.

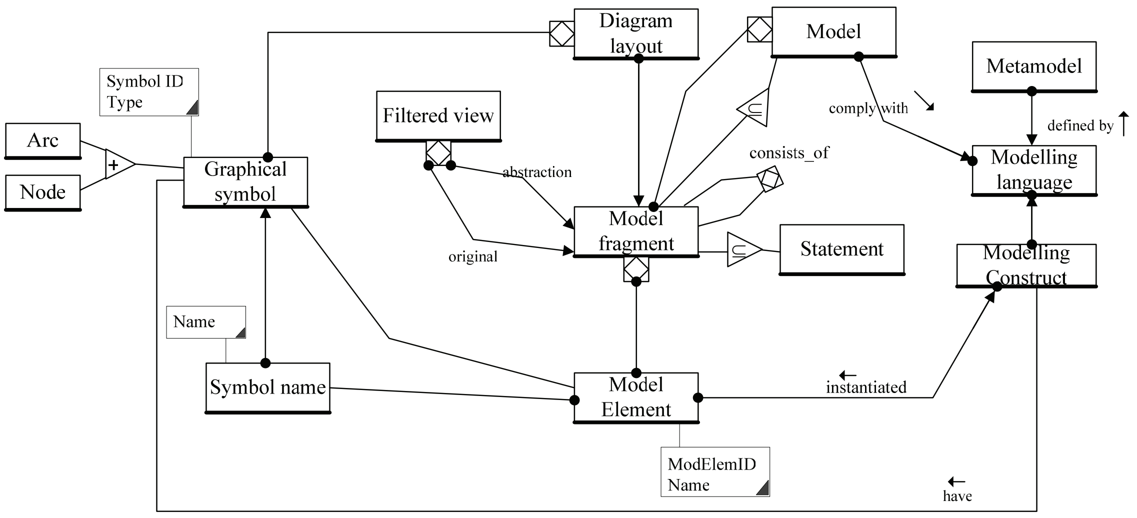


Figure 5.5 Definition of model fragment

Further, Figure 5.5 defines model fragment having layout information, denoted as *diagram layout* – a graphical presentation of a *model fragment*, composed from *graphical symbols*, their *position*. All *diagram layout* have a *model fragment* which graphical layout their represent. *Graphical symbol* and *symbol name* are used to represent *model element*. *Modelling constructs* may have *graphical symbol* to represent them.

A model is a sign system `M=(L, C, R)`, where `L` is a lexicon. The lexicon contains a set of lexical entries for concepts, L_c, and a set of lexical entries for relations, L_r. Their union is the lexicon $L=L_c \bigcup L_r$. `C` - a set of concepts, where for each $c \in C$, there exists at least one statement concerning c in the model. Finally, `R` - a set of relations: a relation `r` ($r \in R$) specifies a pair (`Domain`, `Range`), where `Domain`, `Range` $\in$ `C`.

A model fragment (`MF`) is a subset of the statements in the model `M`: $MF \subseteq M$. While a model element (`ME`) is one of the following: a concept, a relation. Concepts are called atomic elements, while relations are complex elements.

5.3 Internalisation

IS engineering is often viewed as a kind of negotiation process. Different people will use (slightly) different words for the same entities/relations in the same situations (or domains). There are many ways in which the same domain may be described. This issue is recognized, but the answer typically is simple, e.g., "make sure that people involved agree on it". Usually, this is done by finding a domain expert and giving her a power to decide what a domain looks like and how to describe it. Additionally, discussion or negotiation between a small number of stakeholders is often included. Anyway, a uniform domain description is strived for at the end.

It is important to make feasible collaborative real-time modelling. An extensive process of conceptual negotiation and gradual construction of a shared conceptual model is often required to achieve agreement about a domain model. In total, n (n-1) / 2 ways of integration are required for "n" views to be aligned (i.e. Figure 5.6).

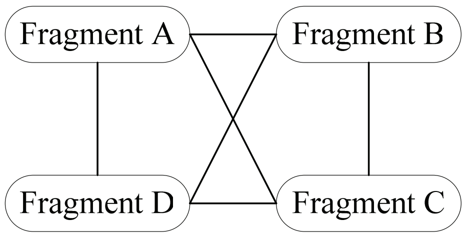


Figure 5.6 Amount of alignment needed

Underlying hypothesis of our approach is that given the same problem domain to reason about, the model developed by different stakeholders will not only differ, but as well will have some overlapping parts, i.e. some parts (views) in different models are commonly shared. In order to integrate the distributed models, these commonalities should be captured.

5.3.1 A method for model fragment management

The method for model fragment management consists of 3 basic steps (see Figure 5.7):

Step_MF1 - Model matching and similarity identification. Model integration typically involves identifying the correspondences between two models, determining differences in definitions, and creating a new model that resolves these differences (see section 5.3.2)

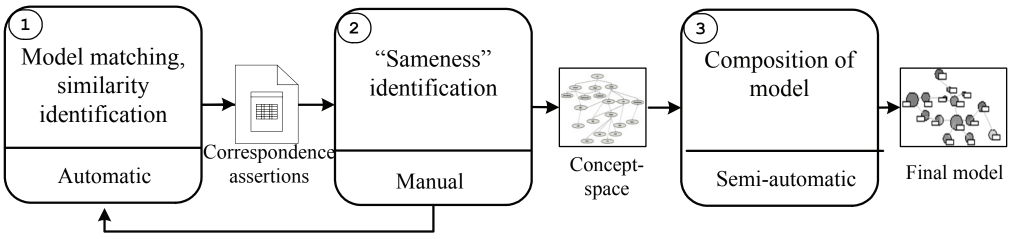

1
Model matching, similarity identification
Automatic
Correspondence assertions
2
"Sameness" identification
Manual
Concept-space
3
Composition of model
Semi-automatic
Final model

Figure 5.7 Model fragment management: functional steps

Step_MF2 - "Sameness" identification. Model fragment owners (authors) are responsible for verifying the mapping results point out the same concepts (see section 5.3.3).

Step_MF3 - Composition of models. In this step model fragments are composed based on "connection points" identified in previous step (see section 5.5).

5.3.2 Similarity Management

A main problem with words is that they may mean more than one thing; or several words may mean the same thing. Since we adopt constructivistic view of the world, we see that concepts equality identification is hardly possible (if at all possible) without actual participation of their creators (modellers).

Here we see similarity identification as a means to relate different model fragments, though we seek for the equivalence (sameness) of the concepts denoted by different developers. Identification of sameness (identical concepts) is not possible without a creator explicitly identifying whether a particular concept specified by her/him does possess an anticipated meaning. Mainly two authors (creators) need to approve or reject identified concept similarity. Approval would mean an establishment of sameness relationship between concepts. Later specifying the proper concept name, still allowing to keep personally preferred term as an alias name for that concept. This is kept in a local version (personal) model fragment.

Concepts descriptions (extensions) are provided by the involved stakeholders. That supports a negotiation process about aligning stakeholders' views and aim to reach a pragmatic agreement about domain conceptualisation. It is noted by Hoppenbrouwers et al. (2005b) that “even if people are willing to and capable of reading models thoroughly, text needs to be added. Models alone never suffice.”

Here we adopt an iMapper system (Su, 2004) for concept mapping (e.g., computing concept similarity). The system is based on computing cosine similarity based on concept feature vectors, constructed from extension of the concepts, i.e. natural language documents. The iMapper system is extended by adopting WordNet electronic lexicon (Miller et al., 1991).

5.3.3 Sameness management

Results of similarity calculation are available to the developers (creators) of model fragments. Then they start negotiation, i.e. clarification of their intentions when specifying the model fragment. The goal is to verify proposed mappings by pointing out the same concepts and achieving agreement about the concept name, if they used different terminology.

These two steps are iterated as many times as new fragments are signed-in to the repository. Identification of “sameness” results in the common knowledge layer or so called “concept-space”, where the commonly agreed concepts and relations between them are placed. This layer is used to differentiate from the local namespace, which is kept unique for each developer allowing to use own vocabulary. This allows maintaining a local scope for the names in order to avoid collisions with the names used by others. I.e. after having identified the concepts being the same, despite of different term used to name them, the “equality” relationship is established between local concepts named ‘bike’ and ‘bicycle’ and the agreed concept named ‘bicycle’.

In a case when authors of different model fragments identify own concepts being the same as colleague’s, the families should not be merged, as each of developer might prefer to use different terminology. I.e., generic object ids (family id) are preserved. The relations to agreed concept are established. For instance, Figure 5.8 depicts above described situation, where two developers have developed alternative models, one used name *A*, while another one preferred to use *A1*. In Figure 5.8 is assumed that they agreed about the concept name being *A*.

Halpin (2001) mentions the occurrence of homonyms in stakeholder interaction, and proposes to approach this problem: “you should get [stakeholders] to agree upon a standard term, and also note any synonyms that they might still want to use” (Halpin, 2001). Actually, it is recommended that lists of homonyms and synonyms from the domain are kept.

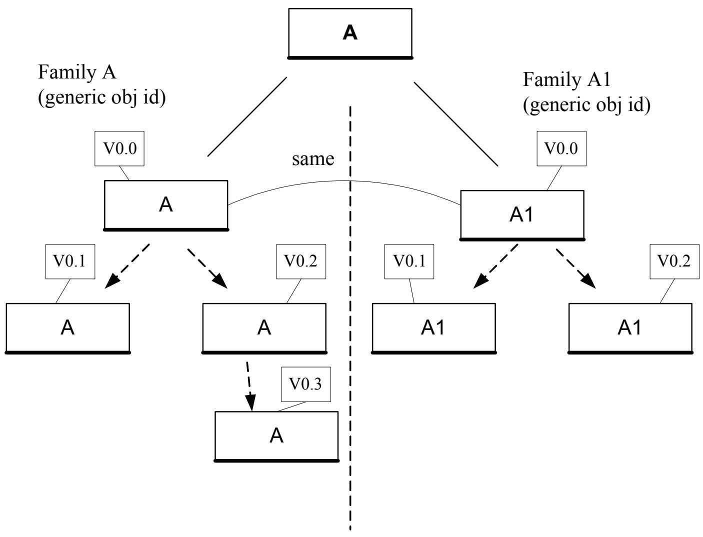


Figure 5.8 Connecting two families

5.4 Refinement of concepts

An analysis of problem starts by forming a mental model of the problem at an abstract level. This model later is refined to a concrete model as more information is obtained (Loucopoulos & Champion, 1988). A domain model should ideally be a product of a shared understanding of domain's stakeholders (Hoppenbrouwers et al., 2005a).

Here, we mainly focus on static (class) diagram which presents concepts and their relationships. Hence, the integration refining issues include abstracting concepts and refining concepts, adding and deleting properties of concepts, adjusting types of properties, abstracting transitive relationships into high level relationships and refining relationships into low level relationships. We define a set of generic actions for the above mentioned refinement transformations. Before formulizing those refining issues, we make some definitions. Let $\mathtt{UoD_I}$ be Universe of Discourse for integrated model, and $\mathtt{UoD_D}$ – Universe of Discourse for particular local model fragments. Then, $\mathtt{C_D}$ is a concept used from a local model fragment and $\mathtt{C_I}$ is the concept in the integrated model. $\mathtt{P(c)}$ is the set of properties of concept $\mathtt{C}$ and $\mathtt{p}$ is a property, $\mathtt{p} \in \mathtt{P}$. While, $\mathtt{R(C_i, C_j)}$ is the relationship between concepts $\mathtt{C_i}$ and $\mathtt{C_j}$.

Action 1. Abstraction of concepts. Concepts used in local models are usually more concrete. Often, during the integration, super concepts are needed to generalise those sub concepts, or even replace sub concepts if the sub concepts are not important in an integrated model.

Let, $\mathtt{C_{Di}}$ and $\mathtt{C_{Dj}}$ be two concepts from a model $\mathtt{i}$ and model $\mathtt{j}$. Both concepts are elements from the same domain (UoD). Then a concept $\mathtt{C_I}$ from the domain of integrated model will be a super-concept of $\mathtt{C_{Di}}$ and $\mathtt{C_{Dj}}$ in the integrated model.

$$\begin{aligned} &C_{Di} \in UoD_I \wedge C_{Dj} \in UoD_I \wedge \exists C_I\ (C_I \in UoD_I \wedge C_{Di} \subseteq C_I \wedge C_{Dj} \subseteq C_I) \\ &\quad \Rightarrow C_I = \text{Abstract}(C_{Di}, C_{Di}) \end{aligned} \qquad \textit{Eq. (5.1)}$$

Action 2. Refinement of concepts. There is a need to create new concepts, when UoD$_I$ of an integrated model is broader than the one considered in the local model fragments. Some of such concepts are created based on a relationship between existing concepts.

$$\begin{aligned} &R(C_{Di}, C_{Dj}) \in UoD_I \wedge \exists C_I (C_I \in UoD_I \wedge C_I \notin UoD_{Di} \wedge C_I \notin UoD_{Dj}) \\ &\quad \Rightarrow \text{Create}(C_I, R(C_{Di}, C_{Dj})) \end{aligned} \qquad \textit{Eq. (5.2)}$$

Action 3. Addition and/or deletion of properties of concepts. Certain properties of concepts are ignored in the distributed model fragments as being not important in a limited scope or in a certain viewpoint, but they might be critical for an integrated model. On the other hand, certain concepts contain too many details which are necessary in some isolated models, but inessential for the integrated system.

$$\exists p (p \in UoD_I \wedge p \notin P(C_D)) \Rightarrow \text{AddProp}(p, C_I) \qquad \textit{Eq. (5.3)}$$

$$\exists p (p \in P(C_D) \wedge p \notin UoD_I) \Rightarrow \text{DelProp}(p, C_D) \qquad \textit{Eq. (5.4)}$$

Action 4. Adjustment of types of properties. Types of properties usually concern implementation oriented aspects, and have little effects on the semantics of models. Meaning that possibly the same property has different types in different models. In order to keep the consistency of integrated model, types of the same property should be unified obeying implementation requirements of system. Let `Sem(p)` be the semantics of property `p` and `T(p)` be the type of property `p`.

$$\text{Sem}(p_{Di}) = \text{Sem}(p_{Dj}) \wedge T(p_{Di}) \neq T(p_{Dj}) \Rightarrow \text{Adjust}(T(p_{Di}), T(p_{Dj})) \qquad \textit{Eq. (5.5)}$$

Action 5. Abstraction of transitive relationships into higher level relationships and refinement of relationships into lower level relationships. A transitive relationship is the semantic equivalent of a collection of normal relationships (Egyed, 2003). The transitive abstraction relationship is the high level relationship and a direct relationship which can not be refined is low level relationship. With different requirements, perhaps only high level relationship is enough while on other cases low level relationship is necessary. There are three generic relationships – generalisation, aggregation and association, which are supported by most modelling languages. The transitive abstraction rules for different combination of three generic relationships are different. In (Egyed & Kruchten, 1999), they developed a set of transitive abstraction rules for inference of transitive relationships (e.g., classA-*association*-> classB<- *aggregation*-classC $\Rightarrow$ classA-*weakAssociation*->classC, meaning that, if classA has association relation with classB, and classB is aggregated into classC, then the resulting abstraction would be weak association between classA and classC), which we do adopt for our purposes. Given the combination of $R(C_{Di}, C_{Dj})$ and $R(C_{Dj}, C_{Dk})$ satisfies one of transitive rules, the result would be $R(C_{Di}, CDk)$, while R here is specified as either generalisation (R_{Ge}), aggregation (R_{Ag}) or association (R_{As}) and parameters are non-transitive.

$$R(C_{Di},C_{Dj}) \cup R(C_{Dj},C_{Dk}) \in \text{RuleSet} \Rightarrow R(C_{Di},C_{Dk}) \qquad \textit{Eq. (5.6)}$$

5.5 Model Composition Management

While the cooperative work support is centred on coordination of the work, the collaborative modelling support is centred on composition of the work. Models change just as the software code does. These changes are caused by changes in the domain itself or in the conceptualisation of the domain (e.g., modeller's knowledge about the domain changes or the domain itself changes). *Conceptual*, *terminological* and *layout* changes should be distinguished in model version control.

Furthermore, model development in large projects is a dynamic process in which multiple developers participate, releasing subsequent versions of a model. Naturally, collaborative development of model requires tools that are similar to code-versioning tools, and different at the same time. A new version of a model is created in one of following ways: a) the stakeholders are adding information to an existing model to make it complete and precise; b) earlier constructed model fragments are included as part of larger model (fragment).

We define configuration being a set of revisions, where each revision comes from a different object family, and the revisions are selected according to a certain criterion.

5.5.1 Composition and manipulation of composition

Views may be overlapping (see Figure 5.9) and language constructs representing the same object may appear in different model fragments. When user changes a term in one view, the change should be propagated to other views as well. Change itself and propagation of the change should result in a new version of the fragments.

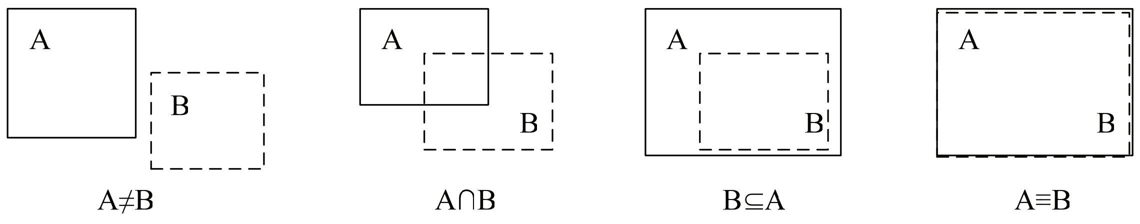


Figure 5.9 Different types of overlap of views

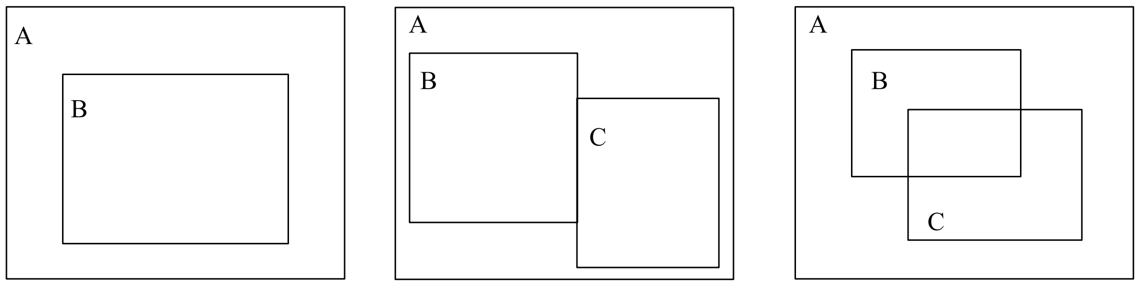


Figure 5.10 Three different inclusions (compositions)

Figure 5.10 illustrates three different compositions of model fragments. They are as follows.

- *Inclusion*. Two different ways are possible: copy or import operations. Copy creates a duplicate a product fragment, i.e. creates a local copy that is a part of new product fragment, i.e. similar to merge. New identity is created, just a origin is preserved for future reference, otherwise it is a new object family. While import uses the object in a composition, whenever imported object is updated, new version of composed object is created automatically, i.e. change of B implies change of A.
- *Connected by the link*. Model fragments B and C have nothing in common, but they are composed into model fragment A by establishing the relation between concepts, e.g., fragment B defining a *kid* being a *person*, fragment C defines *transport means*, including *bicycle*. Then relation *kid.rides* (or *bicyclist*) is established between concepts *bicycle* and *kid*.

$$\exists b[b \in B] \wedge \exists c[c \in C] \wedge [B \subset A] \wedge [C \subset A] \Rightarrow R(b,c) \qquad \textit{Eq. (5.8)}$$

- *Connected using "connection point"*. This is done semi-automatically, after authors have agreed about the sameness of concepts. In this case model fragments have overlapping parts.

5.5.2 Semi automatic configuration

Filters. User can apply *filters* to model fragments to hide and show the concepts of interest. The application of a filter can either modify the model fragment which it is applied to, or create a new model fragment. System offers two kinds of filters: *inheritance* and *derived*. The two inheritance filters make it possible to show the sub-concepts and super-concepts of a particular concept respectively. There are three derived filters: *simple*, *composite*.

Simple. This filter is applied to a concept to show all relations which the concept is involved in, with the exception of the hierarchical decomposition. This filter makes it possible to illustrate the concepts that are in some way related to a specific concept at the same level of a particular abstraction.

Composite. When applied to a concept, this filter shows all the concepts that are components of a particular concept through the aggregation. Moreover, it applies the simple filter to each of these concepts.

Following is a brief description of three filter aspects, as given by (Seltveit, 1994).

Level. A filter is applied to either language- or model-level. A language-level filter operates on the model constructs, while a model-level filter operates on the statements of a model.

Inclusive/Exclusive. A filter may specify constructs to be either *included* in the filtered view from the full view or *excluded* from the filtered view. Since we only apply visual filters operating on the current user view of the model, we may denote these aspects as hide/show respectively. None of the filters we apply actually transform the underlying model. They can only create a configuration or new model fragment.

Determinism and scope of effects. A filter is deterministic if it produces the same filtered view each time it is applied to the same view. The scope of effects of a filter is either local (only affecting constructs within the original view) or global (effects propagating outside the view it operates on). We only apply quite simple filters as a complexity reduction mechanism on a presented view, thus all filters we apply are local to the presented view.

Extend view. When viewing a model fragment, user has the option of retrieving and interactively composing a new model fragment. For instance, given a particular concept (referent) appears in several fragments, user asks to return composition of those fragments based on these concepts.

Figures below illustrate the view extension, where Figure 5.11 and Figure 5.12 are the input and Figure 5.13 is a resulting model fragment, i.e., extended view. Here concepts 'bicycle' and 'person' are "connection point", i.e. they are identified as being similar.

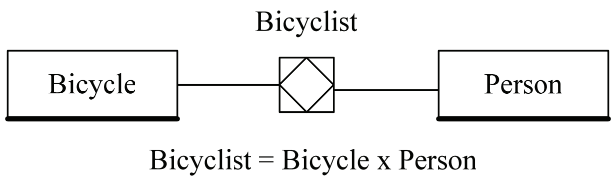


Figure 5.11 Model fragment defining bicyclist concept

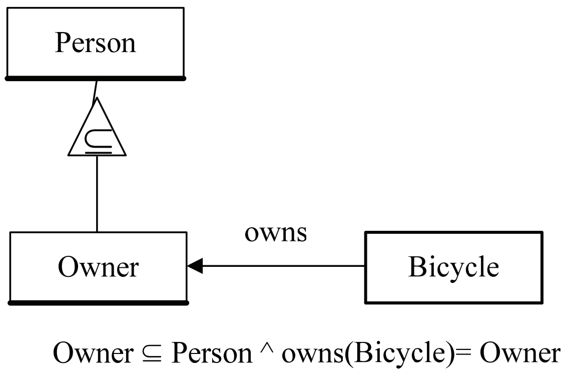


Figure 5.12 Model fragments describing owner relationship

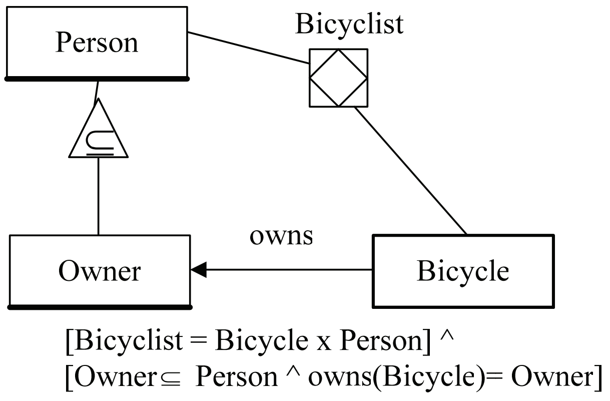

Person
Bicyclist
⊆
owns
Owner
Bicycle
[Bicyclist = Bicycle x Person] ^
[Owner⊆ Person ^ owns(Bicycle)= Owner]

Figure 5.13 Model fragment describing both concepts: bicyclist and owner

This new fragment exists as a configuration of model fragments or may be checked in and stored as a new model fragment.

5.6 Requirements for Model Fragment Management

Based on the above discussion we envision the model fragment management to support a set of requirements as follows.

- *Maintain libraries of model fragments*. Allow uniform access to models in a library, provide pertinent information about each model, such as its authors, domain, etc, provide search capabilities across all the models in a library, allow browsing of the model fragment.

- *Import and reuse model fragment*. Enable users to extend and customize model fragments developed by others.
- *Provide support for model versioning*. Provide mechanisms for storage and identification of different versions of the same model fragment and for highlighting differences between versions.
- *Align and map between models*. Define correspondences between concepts and relations in different models.
- *Merge models*. Given model fragments, create a new composition (model) that incorporates information from all the model fragments.
- *Support automatic update across multiple models*. Based on different ways of composition discussed above.

Figure 5.14 summarises discussion in section 5.2 and denotes six different ways of concept refinement (definition) that should be supported by a modelling environment.

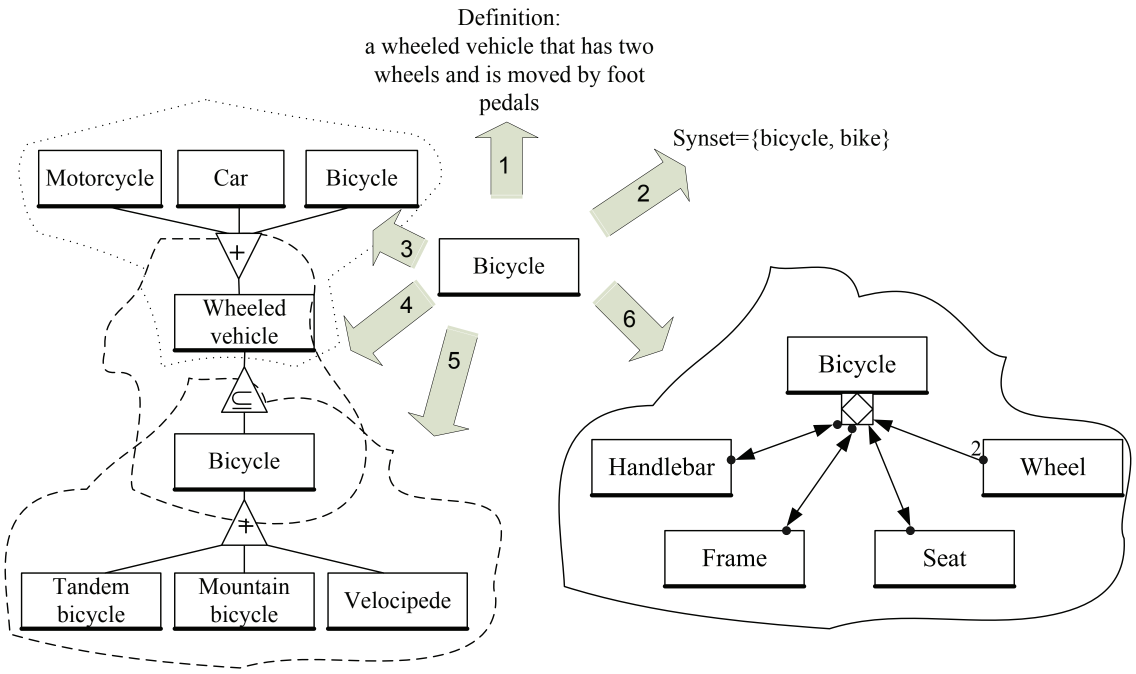


Figure 5.14 Concept refinements

1. Natural language description (definition) of the concept;
2. Synset, i.e. synonyms;

Surrounding concepts:

3. Parent concept and siblings;
4. Just parent concept (hypernym);
5. Sub-concepts (hyponym);
6. Aggregation (if available), i.e. meronyms (part-of).

5.7 Summary

Here we have presented a framework for collaborative distributed modelling. The framework consists of four dimensions, though focus has been put only on three of them. Namely, specification, agreement and commitment dimensions. The fourth is the representation dimension, that is important for distributed modelling, but out of the scope of this thesis.

We argued for the necessity to explicate and define a concept in various ways. That is prerequisite for successful collaborative modelling and further model fragment manipulation. Namely, computation of concepts similarity as a prerequisite for sameness identification and model fragment connection points establishment. Similarity calculation is not a targeted contribution area in this thesis, just adoption of some techniques earlier developed in IS-Group at NTNU. We do not envision any exact matching (i.e., sameness identification) without intervention of authors. It is just a basic mechanism to facilitate an agreement among stakeholders by identifying preliminary similarity.

We have discussed different types of model composition and necessity to automatically update the corresponding composition based on changed component. Before summarising the chapter we have listed main requirements for the modelling environment. The objective of this chapter is to lay down a fundament for implementation of holistic modelling environment for distributed development of IS.

6 Product Fragment Management

"I can't work without a model."
— Vincent van Gogh

This chapter elucidates on basic concepts and ideas behind the framework for product constituent fragments management. It focuses on elaboration of the *step_O2*, *step_O3*, and *step_O4* described in chapter 4.

As we have discussed in chapter 2, one of the main purposes of conceptual model is to serve as a communication tool among participants of the IS development process (Kung & Solvberg, 1986) to help to arrive at a common understanding or agreement on what constitutes the problem domain (Schutte & Rotthowe, 1998). Basic mechanism described in chapter 5 facilitates stakeholders to reach a common understanding on the problem. In successive development phases stakeholders need to exchange the development objects, i.e. communicate their work. Hence, communication is successful if the receiver of communicated information gains the same domain understanding as the sender of the communicated information.

Every communication occurs by some medium. In our case the conceptual model serves as that medium. The sender encodes the message by the medium. Here a product fragment is described by a concept form domain model. The receiver must then decode and interpret the product fragment based on the model.

Conceptual model and ontologies have been named as a tool for bridging the gap between heterogeneous systems. Similarly, the information in databases is only understandable in the context in which it has been created and used, i.e. within a schema. In this chapter we describe the use of a conceptual domain model for encoding the semantics of development objects.

6.1 Domain Model-based Content Management

"Data semantics is the relationship between data and what the data stand for. In order to obtain mutual understanding of interchanged data, the actors have to share a model of what the data represent. Semantic interoperability is about how to achieve such mutual understanding." Solvberg et al. (2002), p. 41

Development of the approach is inspired by a linguistics' method for describing the meaning of objects – the semiotic triangle[8] (Ogden & Richards, 1923). As our approach being user-centric, we base it on the extended semiotic triangle (tetrahedron) by FRSICO (Falkenberg et al., 1997). The extended triangle reflects the constructivistic view. There are real-life objects (referents) observed by a user. The user forms a conception of those observed phenomena. The conception later is represented by a sign (term). This overall subjective construction process is then socialised (see as well discussion in section 2.2.2 and chapter 5) by a subsequent human communication process.

A concept here is referred to as the intention of what one wishes to describe the meaning of. In our method, a concept gives connection between a fragment and a referent, see Figure 6.1. The concept is used to annotate the fragment. The referent is the real world or software world object one wishes to specify or describe the meaning of.

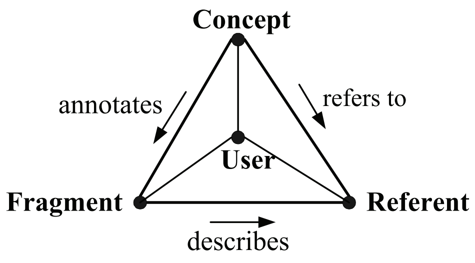


Figure 6.1 Adapted semiotic tetrahedron

The underlying assumption here is that the richer semantic information the product fragment could reveal, the more precise accounts of them could be made and in turn the higher probability that high quality dependency between the fragments will be discovered. The enrichment is conducted by associating each product fragment with the corresponding concepts from a conceptual domain model as in Figure 6.2. Concepts interrelated with each other in the domain model and all the product fragments associated with domain concepts enable us to derive semantic relationships between different fragments. The enriched semantic information is added into metadata to abstract away from heterogeneous representation details and capture information content. A conceptual model is

[8] known as triangle of meaning or Ogden's triangle, as well.

used to interoperate across different representation formats used in the process of systems development. Product fragments association with the concepts from a domain model adds on semantics of the fragment providing a view what part of a problem domain the fragment describes, for instance, a purchase order.

In the domain model, all its concepts are related by generalisation operation, aggregation or other relationships with varying semantics. We assign weights to all those relations according to how strongly concepts are related, based on semantic distance between the concepts. An algorithm for weight computation is discussed in sections 6.2 and 6.3. Meanwhile, the product fragments are linked to concept in the domain model. Relations between fragments and concepts are based on the semantics of the fragments. After having constructed such model and links, the hierarchical position of the concept, semantics of generic relationships between concepts and specific-weighted relations are used to relate heterogeneous fragments and to estimate likelihood of impact from altering one fragment to another. Gradually, fragments linking through domain model (marked by '-!-' in Figure 6.2) are refined to direct (more precise) dependency links between fragments (marked by '-?-' in Figure 6.2). Basically, two kinds of relationships are used in our approach: semantic associations between fragment and concept; and direct relationships between fragments based on dependency between them.

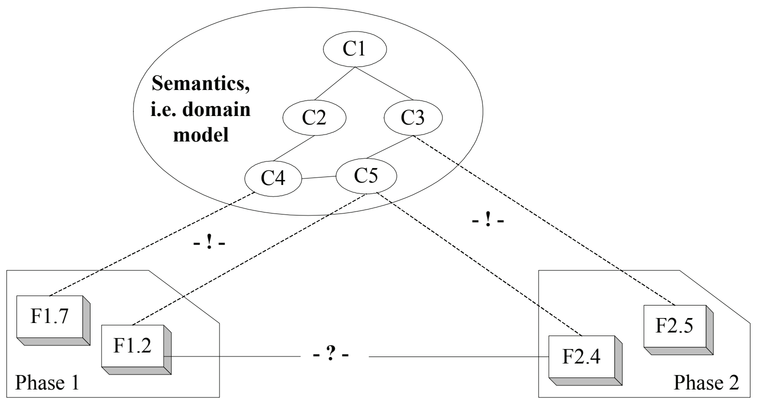


Figure 6.2 Conceptual view of the method

To sum up, two major parts constitutes this part of the method. *First* is the semantic enrichment process where the product fragments are associated with concepts. Developers browse the domain model and interactively associate (classify) product fragments by selecting model fragments (in terms of selecting domain model concepts and named relations) that describe the contents (semantics) of the product fragment. *Second* is the exploitation of enriched semantics (the model itself and association links) to enable impact prediction, and stepwise refinement of those associations to direct dependency links between product fragments, see Figure 6.3. Namely, *product fragments* are direct linked, i.e. *direct dependency* is explicitly denoted among fragments, or connected through *semantic association* to *domain concept*(s).

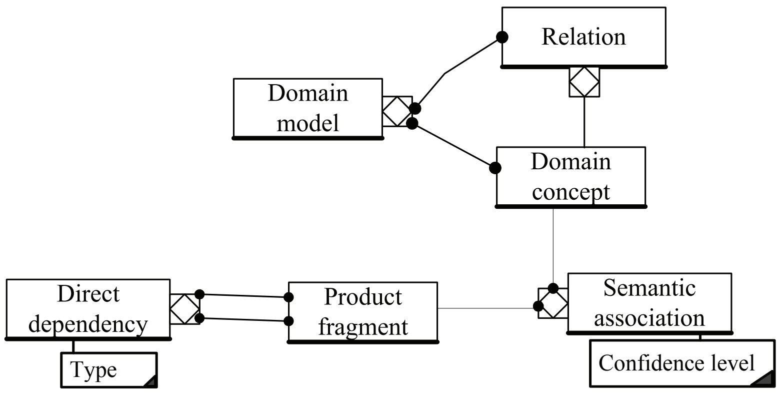


Figure 6.3 Product fragment dependency management

Associations can be seen as paths connecting two fragments. The paths can involve any number of concepts in a domain model. In order to relate product fragments we compute semantic relatedness based on relations between concepts in a conceptual domain model (i.e. concepts similarity) and strength of association between a product fragment and a concept (cluster). Next two sections elaborate on a computational algorithm.

6.2 Semantic relatedness

Here we describe the way we compute semantic relatedness between concepts. These values lay down a fundament to compute relatedness among the product fragments and to estimate change impact. In our method we distinguish three types of relationship in a conceptual domain model: *generalisation* (specification) relationships; *aggregation* (part_of) relationships and *other relationships with varying semantics*. However, these relationships have different semantics and imply distinct semantic distances between domain and range concepts. Weight of the relationship is represented as semantic distance between the concepts. Therefore, the weights for different type of the relationship are computed differently. The following subsections discuss the weight computation for every type of the relationships.

6.2.1 Generalisation relationships

Generalisation relationship between concepts in a conceptual domain model denotes hypernymy relationship between the words, i.e. one concept being more generic than another. For instance, “a vehicle” is hypernym of “train”, “airplane”, and “car”. Concepts placed lower in this type of hierarchy are specialisations of those higher ones in the hierarchy. Figure 6.4 depicts a concept “wheeled vehicle” which is the highest concept in this particular hierarchy, thus, most general. Contrary, concept “velocipede” possesses more precise meaning than “wheeled vehicle” or “bicycle”. It is obvious that a sub-concept (hyponym) is a specialisation of a super-concept (hypernym) and has more precise meaning. Since properties of a super-concept are inherited by a sub-concept, alteration of a product fragment associated with a generic concept will more likely have an impact on fragments associated with specific concepts (sub-concepts) than other way around.

In order to adjust concept weight in a generalisation hierarchy, we define `c_i` being a concept `i` at hierarchical position `HP_i` in a path `P`. A concept weight `cw` is defined as follows.

$$cw_i = \frac{HP_i}{|H|}. \qquad \textit{Eq. (6.1)}$$

Where `|H|` is total height of the hierarchy within path `P`, where the highest concept hierarchical position is equal to 1. For instance, the concept "car" weight in Figure 6.4 is 0.75, as its `HP`=3 and `|H|`=4, whereas, the concept "vehicle" weight in the same hierarchy is 0.25.

A path length is defined as a function of various intermediate weights. We define the path length `pl` in the hierarchy `H` as follows.

$$pl_H = 1 - \left((|c|-1) \times \prod_{i=1}^{|c|} cw_i \right). \qquad \textit{Eq. (6.2)}$$

Where `|c|-1` is the total number of edges between nodes in the hierarchy `H`. To illustrate this, consider the path from "bicycle" to "vehicle"[9] in Figure 6.4:

$$pl_{bicycle-vehicle} = 1 - \left(2 \times \frac{1}{3} \times \frac{2}{3} \times \frac{3}{3} \right) = 1 - 0.44 = 0.56 \cdot$$

While a path length from "scooter" to "bicycle" is equal to 0.72, showing that concepts "bicycle" and "vehicle" are closer semantically than "scooter" and "vehicle".

There are two restrictions for the above introduced calculation of path length. First, a type of specialisation - *overlapping* and *disjoint specialisations*. For instance, semantic gap (distance) between disjoint concepts "car" and "bicycle" are bigger than between "scooter" and "motorbike", which are defined as overlapping concepts. Second, a direction of inheritance. A concept inherits properties of its super-concept, not contrariwise. This restriction is important when computing change impact probability. For instance, if a product fragment specifying requirements for a speed of car (i.e. associated with a concept "car") has been altered, there is less probability that product fragments associated with a concept "wheeled vehicle" will be impacted by this change. Conversely, change of requirement restricting speed of wheeled vehicle most likely will have an impact to the product fragments associated with a concept "car".

[9] Note, that |H| in this case is equal 3.

In order to resolve these two restrictions, we introduce two coefficients with intention to discriminate semantics in the above discussed cases as follows.

$$k_S = \begin{cases} 0.8, & \text{if overlapping} \\ 1.0, & \text{if disjoint} \end{cases}.$$
$$k_D = \begin{cases} 0.5, & \text{is\_a}(a,b) \wedge a \rightarrow b \\ 1.0, & \text{is\_a}(a,b) \wedge b \rightarrow a \end{cases} \qquad \textit{Eq. (6.3)}$$

Where k_S is a coefficient used to adjust weight for a specification type. Namely, it is used to discriminate between overlapping and disjoint generalisation. A coefficient k_D is used to adjust weight based on a direction when traversing generalisation hierarchy, i.e. given is_a relationship between concepts a and b where a is a sub-concept of b, then weight for the path a -> b will be adjusted by a coefficient 0.5.

Finally, total length of a path (semantic distance between concepts related by generalisation/ specification hierarchy) is computed as follows.

$$TPL_H = k_D \times k_S \times pl_H . \qquad \textit{Eq. (6.4)}$$

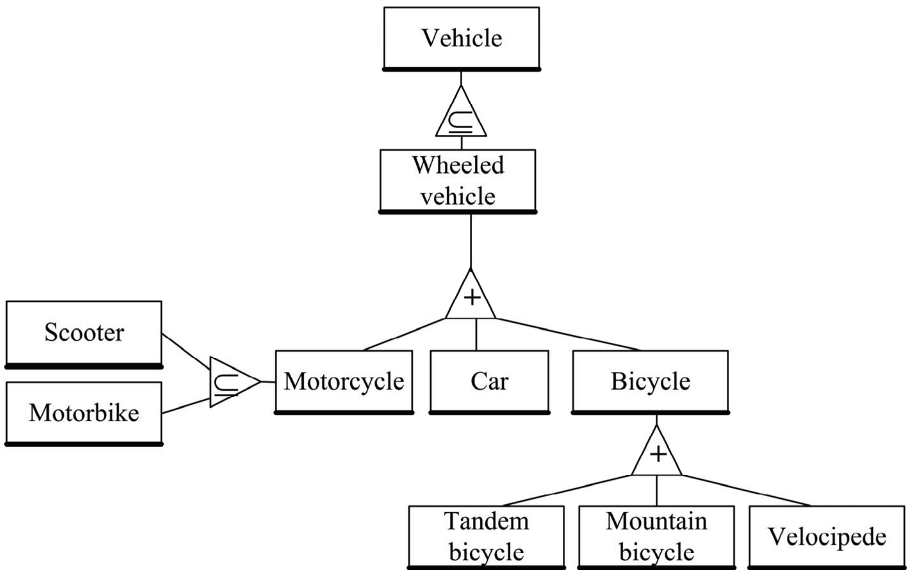
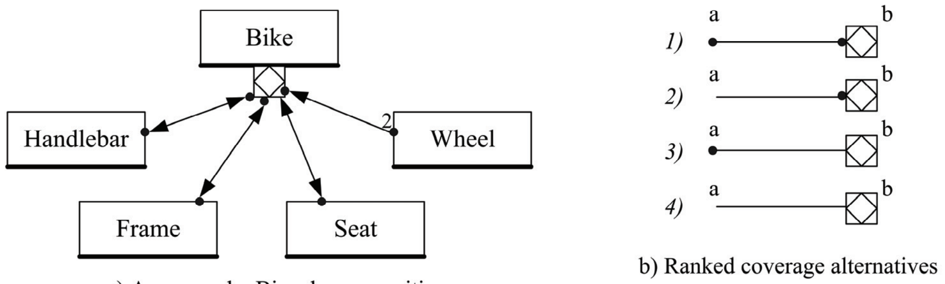


Figure 6.4 Generalisation relationships

6.2.2 Aggregation relationship

Aggregation relates object to the components that make it up via the “part_of” relationships. We treat concepts (components) involved in an aggregation as very close semantically, since they compose a whole. Some components in aggregation are necessary for object to exist. For instance see Figure 6.5 *a)* part, a bike cannot be totally functional without a frame, handlebar and wheels, while a seat is not compulsory though preferable component. Consequently, we assign smallest value equal to 0.1 (closest semantic distance) for the strongest “part_of” relation in the aggregation (see coverage based ranking of “part_of” relations in

Figure 6.5 *b)* part), i.e. full coverage on both ends of the edge. The distance for other relations are increased by a step of 0.05.



a) An example: Bicycle composition

b) Ranked coverage alternatives used in aggregation

Figure 6.5 Importance of aggregation

However, aggregation alone does not capture any functional dependency among components. For instance, there is no information in Figure 6.5 that would indicate that wheels are connected to the frame. This information would even further decrease semantic distance between these two components. Therefore, every direct relationship between elements of aggregation contributes to decreasing the distance by 0.1. Formally, this is defined as follows.

Given concepts `a`, `b`, `c` belonging to a model fragment `MF` and concepts `a` and `b` being components in aggregation of a concept `c`, there is a path between `a` and `c` and a path between `b` and `c`, consequently, there is one path between `a` and `b`.

$$\forall a \forall b \forall c [\exists a \exists b \exists c [a \in \mathrm{MF} \wedge b \in \mathrm{MF} \wedge c \in \mathrm{MF} \wedge [c = a \times b]] \Rightarrow \exists \mathrm{path}(a,c) \wedge \exists \mathrm{path}(b,c)] \Rightarrow \exists! \mathrm{path}(a,b)$$

Consequently, an aggregation path weight `AW` is computed as follows.

$$AW_{path} = \sum_{i=[1,2]} k_i \cdot \qquad \textit{Eq. (6.5)}$$

Where $\mathtt{k_i}$ value depends on importance of a component in an aggregation, i.e. coverage specified on edge as follows, see ranking of coverage in Figure 6.5.

$$k_i = \begin{cases} 0.10, \text{ if } & full\_coverage \times full\_coverage \\ 0.15, \text{ if } & partial\_coverage \times full\_coverage \\ 0.20, \text{ if } & full\_coverage \times partial\_coverage \\ 0.25, \text{ if } & partial\_coverage \times partial\_coverage \end{cases}$$

Given concepts `a`, `b`, `c` belonging to a model fragment `MF` and concepts `a` and `b` being components in aggregation of a concept `c`, and a direct relation between `a` and `b`, then there is a path between `a` and `b`.

$$\forall a \forall b \forall c [\exists a \exists b \exists c [a \in MF \wedge b \in MF \wedge c \in MF \wedge [c = a \times b] \wedge \exists R(a,b)] \Rightarrow \exists path(a,b)]$$

This path between a and b identifies concepts a and b being semantically closer, and a total weight for an aggregation path TAW between them is refined as follows.

$$TAW = AW_{path} - k_{dir\_relation}. \qquad \textit{Eq. (6.6)}$$

Where $k_{dir\_relation}$=0.1 as discussed above.

6.2.3 Other relationships

Above defined computation of a semantic relatedness is based on semantics of relationships in a model. Result of the equations is dependent on correctness of model. For instance, having defined a concept "transport" and having skipped concepts "wheeled vehicle" and "vehicle" would return semantically incorrect value. Though a numerical value is rather relative and is subject for interpretation. For our method, this would not be a major drawback, as all weights might be adjusted manually during ISE process and might be negotiated among stakeholders while modelling.

However, in order to strengthen precision and reliability of computed weights, as well to be able to calculate semantic distance between concepts related by other relationships we have adapted an iMapper approach (Su, 2004) originally created for concept mapping (e.g., computing concept similarity). The system is based on computing cosine similarity based on concept feature vectors, constructed from extension of the concepts, i.e. natural language documents, or textual descriptions provided by modellers or access from the WordNet database.

In order to compute a semantic distance (SD) between concepts having associated textual descriptions (extensions) or linked by other type of relationships we use an equation as follows.

$$SD_{(a,b)} = 1 - sim(C^a, C^b) = 1 - \frac{\overrightarrow{C^a} \times \overrightarrow{C^b}}{|C^a| \times |C^b|} = 1 - \frac{\sum_{i=1}^{n} \left(C_i^a \times C_i^b\right)}{\sqrt{\sum_{i=1}^{n} \left(C_i^a\right)^2} \times \sqrt{\sum_{i=1}^{n} \left(C_i^b\right)^2}}. \qquad \textit{Eq. (6.6)}$$

Where, C^a and C^b are feature vectors for concepts a and b, n is the dimension of the feature vectors, $|C^a|$ and $|C^b|$ are lengths of the two vectors. Result of $sim(C^a, C^b)$ is deducted from 1 in order to convert semantic similarity value to represent semantic distance. Recall that our method is based on semantic distance between concepts and product fragments, where smaller value shows concepts (product fragments) being semantically closer, i.e. contrarily to semantic similarity value, where higher values show concepts being more similar.

6.3 Relatedness and Impact Assessment

In this section we discuss an overall algorithm used for computation of the semantic relatedness of the product fragments, i.e. Dijkstra algorithm for shortest path using Weighted Graphs. The above defined formulas for semantic relatedness computation result in weighted graphs. Before introducing the algorithm and illustrating the overall technique for change impact assessment, we revisit an activity of product fragments association with concepts from a domain model, i.e. *step_O2* briefly described in chapter 4.

6.3.1 Association of product fragments

Product fragments are uploaded to the repository and related to the structure of problem, i.e., they are associated with one or more concepts from a domain model. Developers can specify a confidence level for every association, e.g., specifying the strength of association. For usability sake we use categorical ranges for confidence levels instead of numerical. Three confidence levels are defined. They are as follows. *High* confidence level of association corresponds to numerical value of 0.2; *medium* is represented by 0.5 when computing overall semantic relatedness and *low* confidence level is equal to 0.8.

6.3.2 Weighted Graphs

Having above defined and computed weights, conceptual domain model is processed as weighted graphs. The shortest path algorithm is used to compute which fragments are most likely to be impacted.

G is a weighted graph. The length (or the weight) of a path P is the sum of the weights of the edges of P. That is, if P consists of edges e_0, e_1, …, e_{k-1} then the length of P, denoted L(P), is defined as

$$L(P) = \sum_{i=0}^{k-1} w(e_i) \quad \text{.} \qquad \textit{Eq. (6.7)}$$

The distance from a node v (an altered product fragment) to a node u (a possibly impacted product fragment) in G, denoted d(v, u), is the length of minimum length path from v to u, if such path exists. We calculate a shortest path using Dijkstra algorithm for single-source shortest path (Dijkstra, 1959) from v to each other node in G, treating the weights on the edges as distances.

Once all values are computed, they are normalised to fall into range of [0,1]. Normalised length NL of the path P between two particular product fragments is calculated as follows.

$$NL_j(P) = \frac{L_j(P)}{\max(L(P))} \quad \text{.} \qquad \textit{Eq. (6.8)}$$

Eq. 6.8 returns a semantic distance between two product fragments in a range [0,1]. Smaller value indicates particular product fragments being semantically closer than those with a bigger semantic distance, i.e. value returned by Eq. 6.8.

Algorithm 6.1 presents an overall computation of weights between concepts and product fragments. First, a semantic distance is calculated between every pair of concepts based on Eq. 6.6, i.e. using provided textual descriptions for or associated documents with every concept. If textual resources are not available, then a semantic distance is calculated based on relationship type connecting concepts, i.e. either generalisation hierarchy or aggregation. Manual intervention and specification of weight by hand is necessary when the algorithm is not able to return value for the semantic distance. Later, a path length is computed between an altered product fragment and other associated product fragments (Eq. 6.7), finally values are normalised according to Eq. 6.8.

Algorithm 6.1 Semantic distance computation between concepts.

```
Variables   MF – model fragment;
            C    – a set of concepts in a model fragment MF;
            $TR_c$ – textual resource for concept c (concept extension), where $c \in C$;
            H    – hierarchy of concepts, i.e. concepts related by generalisation/
                   specialisation, where $H \subseteq MF$;
            A    – aggregation relationship;
            PF   – a set of product fragments associated with model fragment.
            PFC– a set of changed product fragments, such as $PFC \subseteq PF$.

Function    $\forall$ a (a $\in$ C):
                $\forall$ b (b $\in$ C):
                  If $\exists$ R(a, b):
                    If $TR_a \neq 0 \wedge TR_b \neq 0$:
                        $W_{(a,b)}$=sim(a,b)          //i.e. Eq.6.6
                    Else if R=H:
                        $W_{(a,b)}=TPL_{(a,b)}$       //i.e. Eq.6.4
                    Else if R=A:
                        $W_{(a,b)}=AW_{(a,b)}$        //i.e. Eq.6.5
                    Else $W_{(a,b)}$=n/a $\wedge$ error_message(“weight for R(a,b) not defined”)
                                                  //manual resolution is needed

            $\forall$ pfc (pfc $\in$ PFC):
                $\forall$ pf (pf $\in$ PF):
                  Return L(pfc,pf)                //i.e. Eq.6.7

            Find max(L)
            $\forall$ l (l $\in$ L):
                Return nl=l/max(L)                //i.e. Eq.6.8
```

6.4 Impact Notification and Direct Dependency Association

Impact notification is based on the normalised values of the path length between product fragments as described in the previous section. In order to control amount of notifications sent to stakeholders, we introduce a procedure of sequential impact notification based on dependencies between *phase products*. The procedure is discussed in next sub-section. Direct dependency association based on statistics of impact notifications is discussed after.

6.4.1 Impact notification

Though all the related product fragments are associated with the same domain model concept, we still need an order of notification. Too avoid too many notifications we define interdependencies among the phase products (as discussed in chapter 4). For instance, if there is a change in the implemented code, which changes something in the design, this does not necessitate a change in the requirements. Thus, as explained in Figure 6.6, the requirement fragments that are related to the changed code fragments do not necessarily have to be investigated for possible impact, since the requirement fragments directly influence the fragments of design and test scenario.

Figure 6.6 illustrates configured impact notification sequences and directions (denoted by a solid line). Whereas, dotted lines illustrate a scenario as follows. Consider an altered requirements statement (a fragment of requirements specification). First round of notification will include only test scenarios and design fragments. Second, if impact on design fragment will be confirmed, then algorithm will proceed further, i.e. providing notification about possible impacts on code fragments, and so far.

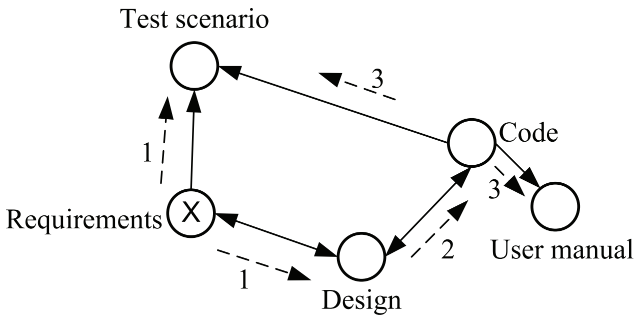


Figure 6.6 Notification sequence of possible change propagation scenario

Note, that alteration of user manual according the notification scheme in Figure 6.6 will have impact only on related user manuals (self-dependency relationship is excluded for the figure). In this way the amount of generated change impact notifications is reduced.

6.4.2 Direct dependency association

As discussed in chapter 4, every change impact needs to be investigated. Every change notification is verified and either confirmed or rejected. This decision is logged. Algorithm 6.2 is used to compute candidates for direct dependency linking. For each pair consisting of one altered product fragment and one possibly impacted product fragments an amount of confirmations and rejections is counted. If impact notifications for a particular pair of product fragments has been confirmed at least 3, then a ration between confirmed and rejected notifications is controlled. I.e., the amount of confirmations should be at least twice bigger than the amount of rejected impact notifications for a particular pair of product fragments.

Algorithm 6.2 Statistical analysis of the log of change impact notifications

Variables	PFC – set of changed product fragments; PFI – set of impacted product fragments.
Function	$\forall$ pfc (pfc $\in$ PFC) in impact_history: $\forall$ pfi (pfi $\in$ PFI) in impact_history: C_{pfi} = Count how many pfi are confirmed R_{pfi} = Count how many pfi are rejected If $C_{pfi} > 3$ and $V_{pfi}/R_{pfi} > 2$: $\exists$ direct_dependency(pfc, pfi)

6.5 Evolution of Domain and Change of Domain Model

It is acknowledged that product development is a wicked problem, where the problem itself is never completely defined before its solution is developed (Solvberg & Kung, 1993). Here we define straightforward scenarios to control changes in domain model. These strategies deal with re-associating the product fragments because of changed concepts.

When having associated product fragments, deletion of the concept in domain model will cause a need to re-associate product fragments. This is done either by automatically associating product fragments to more general concept and lowering the confidence level of a particular association. This scenario is applied only to concepts related by generalisation or aggregation relations. Other scenario is to keep a structure of the domain model untouched, just freezing the deleted concept, not allowing to associate new product fragments.

6.6 Application Scenarios

Here we discuss additional application scenarios for the method. The intention here is to indicate areas where the method is useful.

Maintenance. Given associated product fragments to a conceptual domain model, we able to more efficiently estimate an impact of any new requirement. Since it is easier to associate a new item with a domain concept and get an overview of possible impacts, then investigate a huge set of product fragments for possible dependencies. Amount of concepts always will be smaller than amount of product fragments, even if a domain model is huge and contains thousands of concepts.

Control of the project. Amount of product fragments (recently) associated with particular concepts identifies where most of work is undergoing, i.e. what part of problem has a focus (currently). This allows project manager to distribute resources more equally or prioritise other parts. Project manager uses domain model as a "map" for planning and controlling a project. We define a metric to calculate a focus of a concept `Ci`. The focus is defined as the number of product fragments (`PF`) associated with a concept `Ci` in a domain model compared to the total number of product fragments associated with a domain model.

$$F_i = \frac{|C_i(PF)|}{|PF|} \qquad \textit{Eq. (6.9)}$$

Control of solution boundaries. Consider a case with a broad domain and a correspondingly big domain model. In such a situation it might be decided to computerise only a part of problem domain. Then if product fragments are associated with domain model concepts outside the designated solution area (see Figure 6.7) – it is a sign that solution might be outside the problem area.

$$\text{Comp\_system\_model} \subset \text{Domain\_model},$$
$$\text{c} \in \text{Domain\_model},$$
$$\text{pf} \in \text{PF},$$
$$\forall\, \text{pf}\left[\exists \text{association(pf,c)} \wedge \text{c} \notin \text{Comp\_system\_model} \Rightarrow \text{pf} \notin \text{Comp\_system\_model}\right].$$

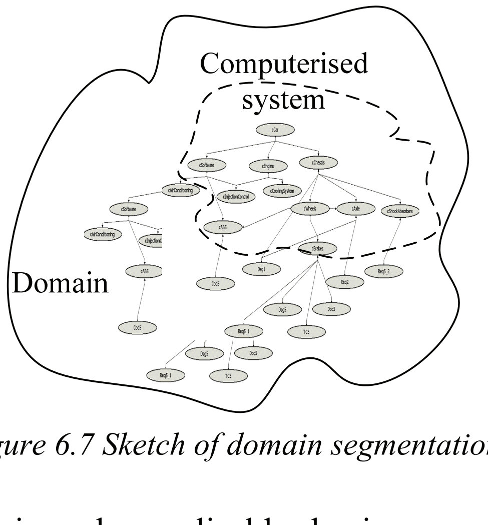


Figure 6.7 Sketch of domain segmentation

Though, this scenario is only applicable having a robust mechanism for automatic association of product fragments to concepts. This issue is further elaborated in chapter 9 under discussion about future improvements.

6.7 Summary

In this chapter we have discussed use of conceptual domain model for explicating semantics of product fragments by associating the product fragment to corresponding concept (cluster). Then based on these associations and intra model relation (model structure) we calculate semantic relatedness of associated product fragments. The relatedness is calculated using Dijksta's shortest path algorithm.

We are dealing with two generic relationship types, i.e., direct (explicit) and associations (implicit) dependency relations. The latter (implicit) are stepwise refined to the former (explicit) by the use of accumulated impact history.

Here we propose to use conceptual models not only to guide the design of a system, but also to actually access and manage information produced during IS engineering. Semantic associations of development objects with a concept from a domain model are intended to communicate the meaning of development objects between stakeholders.

All weights and the way to compute them need to be validated empirically. Most likely, they will need to be tuned for specific settings. That is one of the future works.

7 Realisation of the Method

"There comes a time when one must stop suggesting and evaluating new solutions, and get on with the job of analyzing and finally implementing one pretty good solution."

- Robert Machol

The method, described in the previous chapters, has been applied in a prototype. The prototype has been implemented to verify whether the earlier described method is applicable solution. In this chapter we will outline architecture and design of the implemented prototype. The discussion is focused on functionality specification rather than technical details.

The chapter is organised as follows. First the brief account is given to the environment in which our work has been situated, namely describing the relevant earlier built components. Second, the architecture of our implementation is outlined and discussed. Third, design and main functionality are overviewed.

7.1 Components

An implementation of our method builds on the components earlier created in IS-group. Here we discuss how they are related to realisation of our method and implemented prototype CO_2SY[10] (COOperative SYstem).

The main constituent parts for our implementation environment are illustrated in Figure 7.1 and are discussed in details below the figure. Briefly, the modelling environment is used to model domain, in particular RML editor has

[10] pronounced as 'cosy'.

been used for the domain modelling part of the method. Models are used by iMapper and CnS client for their own purposes, see respectively Su (2004) and Brasethvik (2004). In our method iMapper is used to calculate initial mappings between different views (model fragments) as described in chapter 5. CO_2SY repository system is made to store necessary information for internal purposes of both, iMapper and CnS systems, i.e. model fragments, concepts extension (classified documents), mapping results and lexicon. Output of these systems are stored and interchanged in XML format. Next, the components are elaborated in detail.

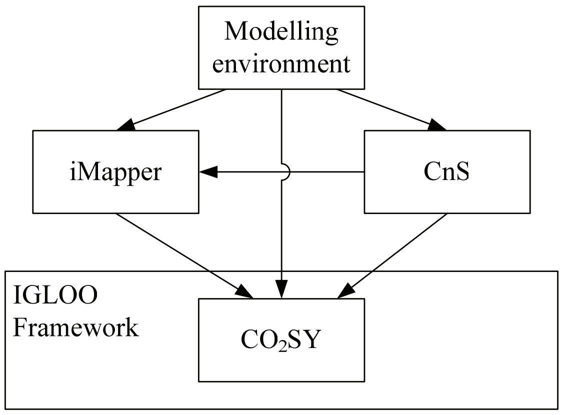


Figure 7.1 Components of method realisation

7.1.1 The IGLOO Framework

The IGLOO framework for cooperative product development (Farshchian, 2001) is defined as an "operating system" for cooperative support. IGLOO can be used for integrating already existing tools into a coherent cooperative environment. IGLOO offers shared workspaces, both synchronous and asynchronous awareness mechanisms as well as annotation mechanisms. IGLOO framework for cooperative work has three basic service layers. A short description of each layer and the types of their services follows below.

Product Layer is in charge of maintaining a shared product space. Shared product space is a virtual space where a group of cooperating users can make available and share their product fragments. Product Layer provides services for inserting new product fragments into the shared product space, for modifying the fragments in different ways, and for creating arbitrary relations among the development objects. Actually, product layer is not meant for storing and processing product fragments, but merely for sharing them. This means that the users decide how much and what aspects of a product fragment they want to share (normally those aspects that are necessary for the cooperation). The relations among the product fragments are generic relations that can be specialized into product-specific relations (such as part-of or dependency relations) or subscription for notification of certain events. Product layer supports opportunistic communication among awareness producers and consumers.

Cluster Layer is the intermediate level between a large shared product space and small groups interacting with this space. Clusters supply centers of interaction with shared product information. Clusters are user-defined collections of development objects from the shared product space (product layer) that are considered by a group of users to be important for performing a task. Cluster Layer allows its users to create clusters, and to customize the clusters' contents (i.e., which objects that are part of the cluster) and form (i.e., how the objects should be represented).

In addition, cluster layer allows a group of users to share a cluster and its content, and to have access to product awareness that is generated by product layer. Clusters are created in a way to provide both, focus (by selecting only a subset of existing product fragments from the shared product space and hiding the other fragments) and overview (by allowing the users to monitor product fragments external to a cluster).

Workspace Layer provides a medium for informal cooperation in small groups of users working on a focused task. A shared workspace provides the medium for a centre of interaction. Each shared workspace in IGLOO may consist of any number of clusters. A shared workspace has in addition access to the shared product space through the clusters within the workspace. In this way the underlying shared product space is used as a unifying component among all the shared workspaces. Each shared workspace can decide to visualize the clusters and the development objects from the shared product space in different forms.

Unfortunately, implementation of the IGLOO framework did not persist. Therefore, we had to adopt and build CO$_2$SY based on the underlying ideas of theoretical IGLOO framework (Farshchian, 2001) in order to support cooperation among developers using CO$_2$SY in a distributed development.

7.1.2 The Modelling Environment

During this work, it has been natural to incorporate into our implementation and reflect on previous modelling methodologies and tools accumulated in the IS group at IDI, NTNU. The main tool in this work is the RML editor. The RML language is a modelling language that initially springed out from the PPP integrated modelling environment (Gulla et al., 1991). PPP initially contained support for several modelling languages; a Process Model Language PrM, an extended ER modelling language (ONE-R) and a rule modelling language (PLD), and also comprised specifications and partial implementations of extensive methodology support; versioning mechanisms (Andersen, 1994), view generation (Seltveit, 1994), concepts and notation for hierarchical modelling (Sindre, 1990), prototyping and execution (Willumsen, 1993) as well as explanation generation and translation of models (Gulla, 1993). Later work have refined the initial modelling languages and also added new languages. The most recent are the RML concept modelling language (Solvberg, 1999), the APM workflow modelling language (Carlsen, 1997), and the task and dialogue modelling languages for user interface design (Traetteberg, 2002).

For model creation we have used the older version of RML editor, RefEdit[11]. Though, a new editor has been implemented by Fidjestol (2005) incorporating some of the ideas discussed in chapter 6 with a purpose to create a platform independent tool (in wxPython). Unfortunately, because of limited time resources it has not reached maturity and remained in pre-prototype quality.

7.1.3 The iMapper System

The *iMapper System* (Su, 2004) has been developed to compute similarity of concepts from different model fragments based on concepts extensions and instances, e.g., natural language documents describing a particular concept. The similarity computation is based on constructing feature vectors and computing the mappings. The prototype of iMapper was developed as a stand alone java application that communicates with the other components through XML file exchange (Su, 2004). We use iMapper as a component for our model fragment management method to compute initial similarity of model fragments, as discussed in chapter 5. In addition, CO_2SY repository system supplements iMapper by providing storage and maintenance of model fragments, concept extensions and results of mapping process.

7.1.4 The CnS Client

The *CnS Client* (Brasethvik, 2004) has been created for the purpose of classification and search of documents using domain model. Additional features include personalisation of model fragments (as a query expression). CO_2SY repository provides storage and manipulation mechanism for model fragments, documents and their classification information. Furthermore, maintenance of lexicon (synonyms) is important for the CnS system for querying purposes as well as for the our method for terminology alignment as discussed in chapter 5.

7.2 Architecture

In order to support the wide range of tools used in a distributed development (VA Software, 2004) we have chosen to develop a repository system communicating with clients by XML-RPC protocol. An overview of CO_2SY architecture is shown in Figure 7.2.

We have implemented a graphical user interface (*GUI*) for repository access and content manipulation at the client side. CO_2SY *GUI* should have integrated modelling environment, as denoted by dotted line between *Modelling Tool* and *GUI* in Figure 7.2. A tight integration of modelling environment and

[11] http://www.idi.ntnu.no/~ppp.

repository has been investigated and implemented by Fidjestol (2005). However, because of limited time resources the implementation by Fidjestol (2005) has not reached a maturity level required for integration with CO_2SY. Section 7.4 is dedicated for more detail overview of implemented functionality from the perspective of user interface.

Above described *iMapper* and *CnS* systems may be configured to connect to the repository through *Model Manager*. Meanwhile, the outputs from all other tools used in a development are checked-in to the repository through CO_2SY *GUI*.

The server side consists of an object-relational database management system that provides a physical, persistent datastore for the product fragments. The repository is implemented using PostgreSQL ORDBMS (Postgresql, 2005). The files can be stored on the file system and can only be accessed through the repository. The data sets could either be stored in a flat file system or in a database. For a large number of data sets, however, storage in databases is preferable because querying and retrieving is more efficient when compared to using a flat file system. Other server side components depicted in Figure 7.2 are as follows.

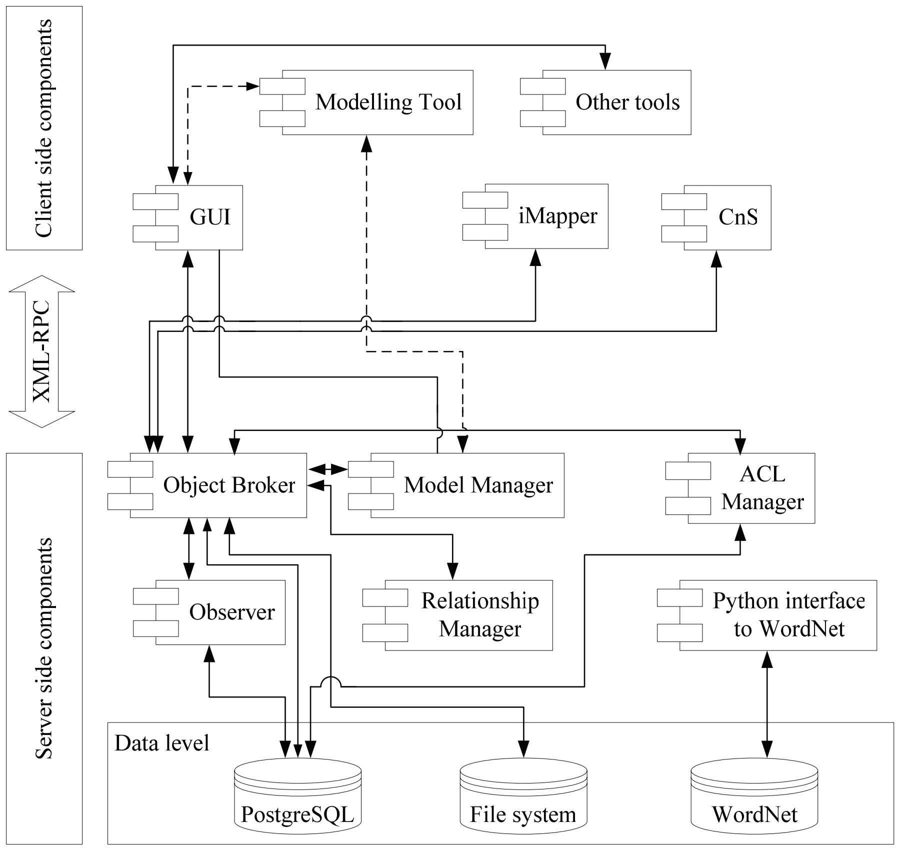


Figure 7.2 CO_2SY architecture

As modelling is an important part of our method and ISE, in general, *Model Manager* has a central role in CO_2SY. Though the role of *Model Manager* is somehow duplicated with *Object Broker*. That is done to better support modelling task and is intended for an integrated modelling environment. The dotted line between *Model Manager* and *Modelling Tool* depicts the intended situation. While current implementation of CO_2SY is based on XML file interchange between earlier version of RML editor (i.e. RefEdit) and *GUI*. If a product fragment is a RML model, it is passed to Model Manager to extract conceptual information and store it in database. Otherwise, there is an *Object Broker* that transforms structured and semi-structured files into repository format based on an object type.

Python Interface to Wordnet. PyWordNet[12] (Steele, 2004) has been used to access the WordNet database to find related terms through lexical relationships in WordNet (i.e., synonyms, hypernyms, hyponyms, and meronyms) in order to support modeling activity and reconcile the model fragments.

Observer module used for a change notification mechanism, which reacts spontaneously to the change of a development object. It can be configured to notify only about the actually performed changes based on uploaded revisions of development objects or include as well future changes, i.e., based on the development objects checked-out from repository.

Observer provides a mechanism to monitor repository modifications. Listeners are activated on a certain events:

- when development objects are created in repository;
- when new development object type is defined in repository;
- when development objects are updated;
- when development objects are suspected to be impacted;
- when links (both, direct dependency and associations with domain model) are established;
- when user logs in/out.

Relationship Manager deals with associations and links among development objects (version graph, inclusion (part_of), and direct dependency links), recall discussion in earlier chapters. *ACL manager* authorizes access to the system and maintains session.

7.3 Design

The prototype is implemented in Python v2.3 (wxPython v2.6 used for graphical user interface) and uses an object-relational database managements system to support repository activities. For this reason, the object relational database, PostgreSQL (2005) is chosen from a group of possible products such as Oracle™

[12] http://sourceforge.net/projects/pywordnet

(2005), MS Access™ (2003), MySQL (2005), to name just a few but prominent DBMSs. The reason to decide on PostgreSQL is that it is a public domain code, i.e. free of charge, that it provides sufficient features that is needed. For instance, PostgreSQL has triggers, stored procedures, and a rich set of built-in functions. In addition, PostgreSQL's procedures and triggers can be written in other languages as well, such as PL/TCL, PL/perl, and PL/python. These additional languages come in two basic flavours, safe and unsafe. Safe allows only for use of things in the programming language that don't affect the host system negatively, such as direct access to the file system.

Next we will present and discuss design decisions regarding earlier stated problems. Figure 7.3 shows a fragment of repository schema used to store results of concept similarity computation. Furthermore, a lexicon for particular concept (both, relational and class concept) is stored with a reference to agreed proper concept name, see left side of Figure 7.3.

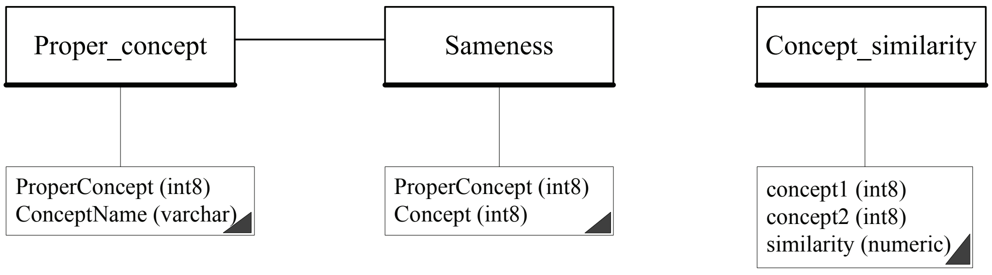


Figure 7.3 Tables for concept similarity and sameness

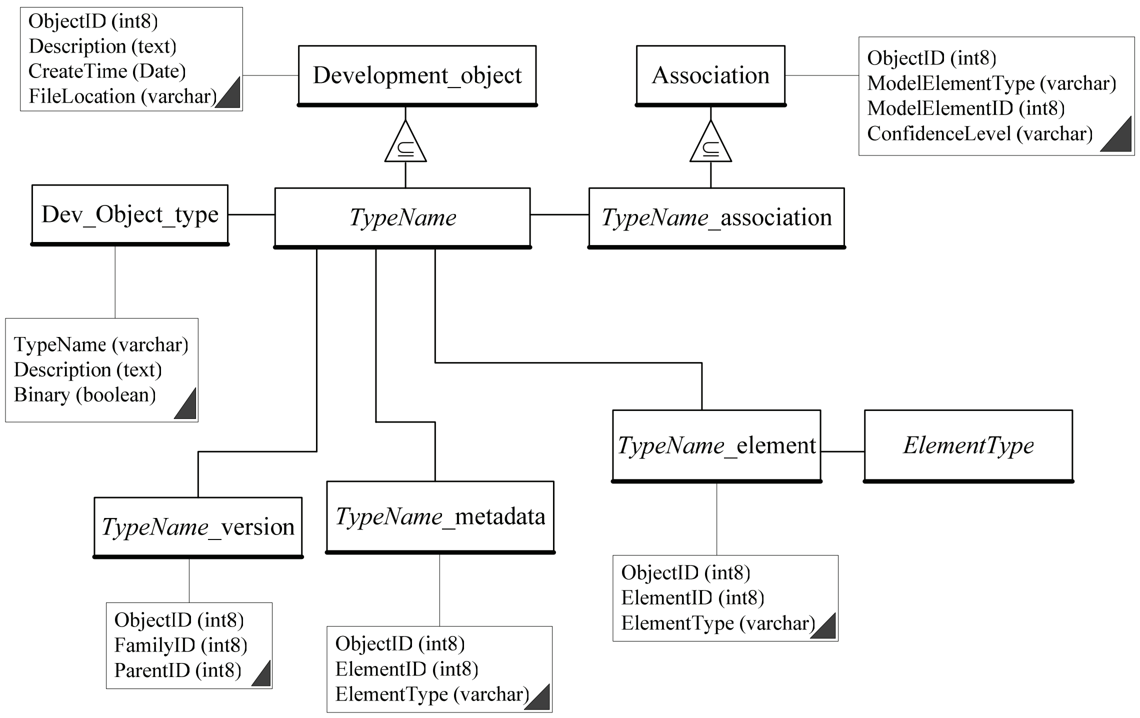


Figure 7.4 Tables to store development objects

As it was discussed in chapter 4, the major design objective for the repository is to make the schema resilient to accommodate different new development object types. Figure 7.4 illustrate tables used as a framework to satisfy that requirement. Table `Dev_Object_type` is used as a register for types of development objects used in a particular project. Where field `Binary` identifies whether a particular object is stored as a binary or as structured development object, i.e. object is structurally parsed before storing in the repository. While entry in field `TypeName` is used to instantiate new tables for a particular object type called *TypeName*. Note, TypeName in italics in Figure 7.4, this is "replaced" by an actual object type name. For instance, having Python_code as an object type tables specified with TypeName in italics in Figure 7.4 will be called as follows. Python_code; Python_code_association (if it is defined as non-binary object type); Python_code_element; Python_code_version; Python_code_metadata.

Table *`TypeName`*`_version` stores object version identification as a triple discussed in Chapter 4. In table *`TypeName`*`_element` it is defined what structural elements we are interested to store for a particular object type, e.g. for python code it would be classes and function, while for RML model we have listed `class_concept`, `individual_concept`, `attribute`, `generalization-subset`, `generalization-element`, `relation_concept`, see Figure A.2 in Appendix A). All specific object type tables inherit from table `Development_object`. The same procedure is done with table `Association`, used to store product fragments associations with domain concepts. Inheritance of the tables is used to optimize querying, especially during a procedure of change impact assessment.

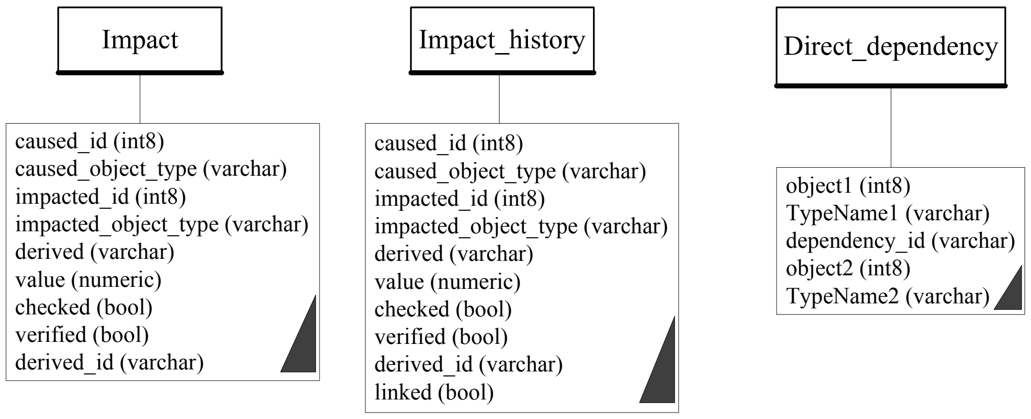


Figure 7.5 Change impact notifications, the log and direct dependency

Figure 7.5 illustrates tables used to store change impact notifications and to specify direct dependency links between objects. Table `Impact_history` is used as a log of decisions taken regarding every change impact notification. The log is further used for computation of statistics to be used for direct dependency links specification, as described in Chapter 6. A screenshot of graphical user interface implemented for this part of the method is shown in Figure 7.7.

7.4 Functionality and User Interface

As it has mentioned earlier the main interface to repository content and management of the product fragments has been implemented in wxPython[13]. Figure 7.6 shows main four interface components. Namely, functionality tabs, information display and manipulation, user identification (login form) and discussion area (chat). It is possible to attach multiple chats to a development objects (or fine-grained elements of development object, if objects are structured) stored in the system. This later is used as textual resource describing a particular development object or concept.

The content of information and manipulation panel depends on a particular functionality tab. Appendix B discusses and visualizes the main functionality of the implemented prototype, illustrating "*B area*" (see Figure 7.6) of particular functionality. While below we briefly overview a fragment of graphical user interface implemented for to manage change impact notifications.

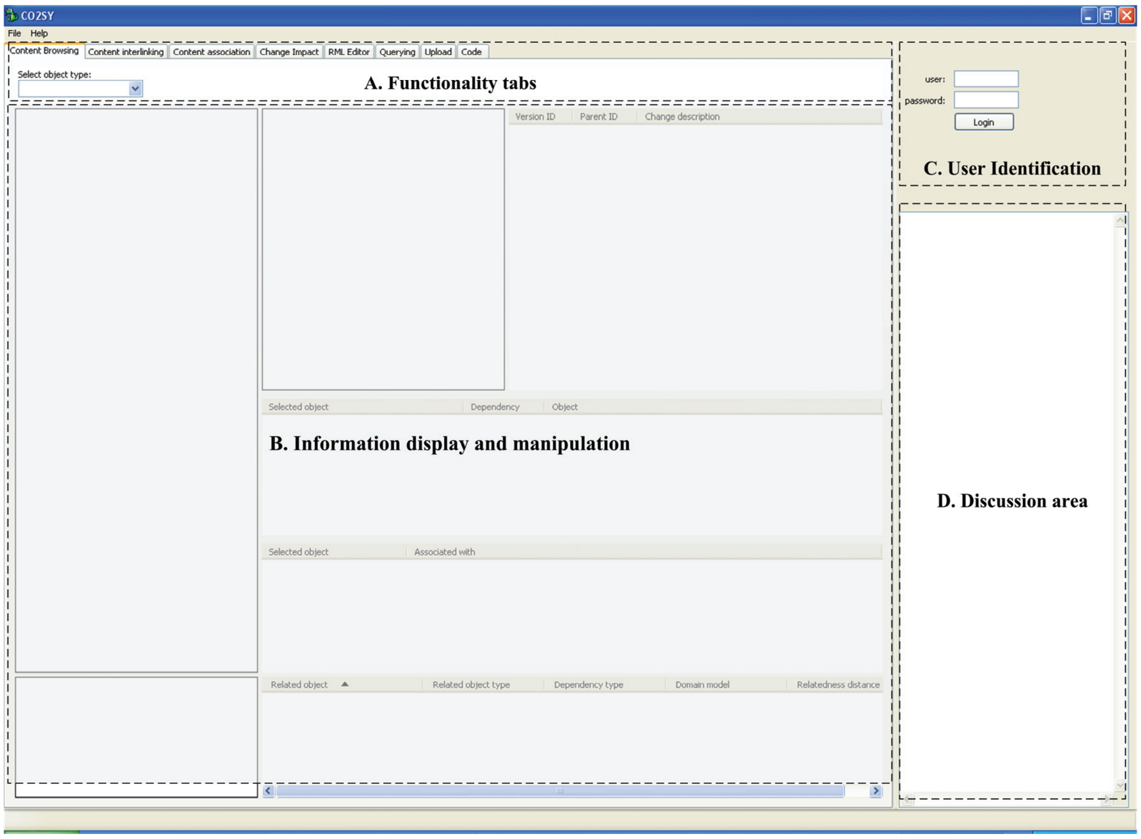


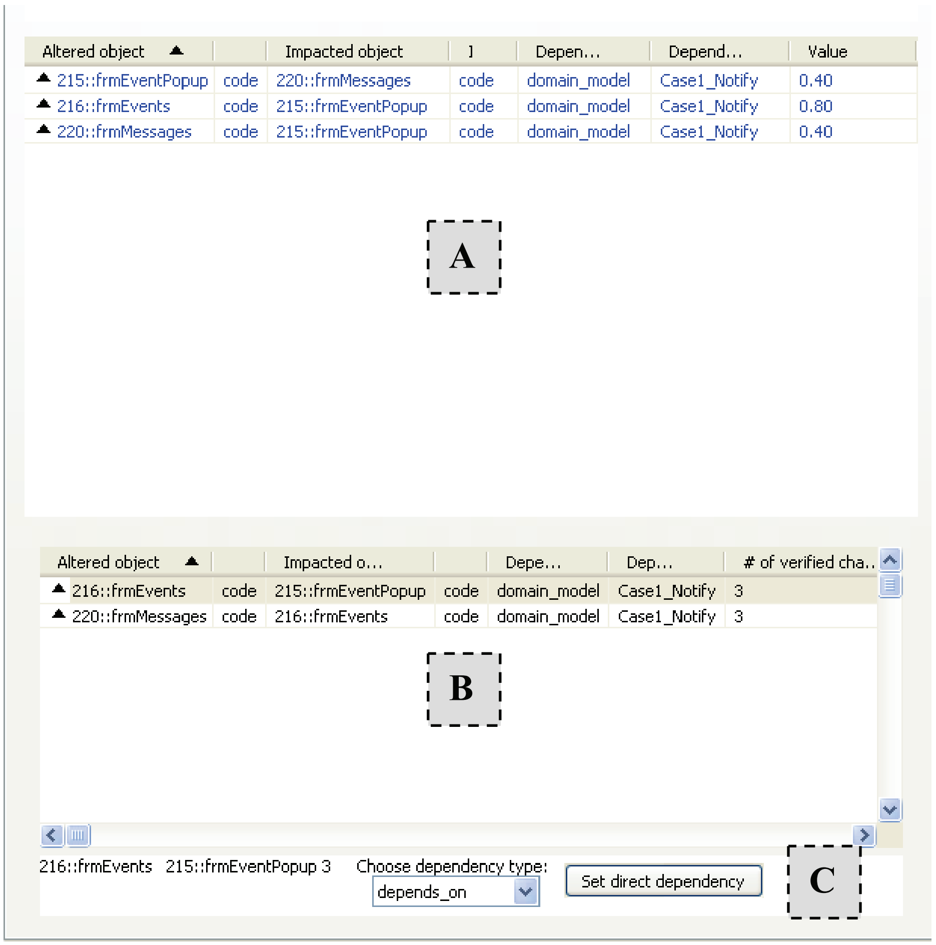
Figure 7.6 Overview of main components of interface

[13] http://www.wxpython.org

Figure 7.7 illustrates the main result window of our method, i.e. impact assessment. The main components of this window are as follows.

`A` area lists all impact assessments. Here are results from both direct dependency links, if any, and based on associations with concepts from domain model. The following information is provided here: *Altered object*, *altered object type*, *impacted object*, *impacted object type*, *dependency* (i.e. whether direct dependency link or domain model based impact assessment), *dependency type* (i.e. either domain model name, since it is possible to associate product fragments with more than one domain model, or direct dependency type), *value* (i.e. semantic distance between two particular fragments).

`B` area lists candidates for direct dependency links, recall algorithm 6.2 described in chapter 6. First six columns are as for the list in `A` area. Last column shows how many times change impact for a particular pair of product fragments has been confirmed. Recall, that here are listed only those pairs which has “passed” algorithm 6.2 computation.



Altered object ▲		Impacted object	I	Depen...	Depend...	Value
▲ 215::frmEventPopup	code	220::frmMessages	code	domain_model	Case1_Notify	0.40
▲ 216::frmEvents	code	215::frmEventPopup	code	domain_model	Case1_Notify	0.80
▲ 220::frmMessages	code	215::frmEventPopup	code	domain_model	Case1_Notify	0.40

Altered object ▲		Impacted o...		Depe...	Dep...	# of verified cha..
▲ 216::frmEvents	code	215::frmEventPopup	code	domain_model	Case1_Notify	3
▲ 220::frmMessages	code	216::frmEvents	code	domain_model	Case1_Notify	3

216::frmEvents 215::frmEventPopup 3 Choose dependency type: depends_on Set direct dependency

Figure 7.7 Change impact management

C area is a form for establishment of direct dependency links. Here end-user can specify the type of direct dependency link between a particular pair of development objects.

7.5 Summary

In the chapter, we have elaborated realisation of the prototype system CO_2SY. First, we have discussed the environment in which our prototype system CO_2SY is developed. Namely, IGLOO framework for collaborative work support, the modelling environment, iMapper and CnS systems. Second, we have elaborated on architecture and design of the prototype. Before summarising the chapter a short overview of main functionality of the implemented prototype is presented.

The implementation is of prototype quality and we have tried to integrate available tools into the system. Unfortunately, because of limited resources it was not possible to integrate with the new version of RML editor. This has an implication to the evaluation of the prototype and method. The evaluation is discussed in the next chapter.

8
Evaluation of the Method

"Qui nimium probat, nihil probat"
Latin proverb[14]

This chapter presents the evaluation of our method. As described earlier, we have specified a method for change impact assessment using a conceptual domain model. Here, the aim is to test the effect and usability of the domain model driven change impact assessment method. An experimental case study is conducted using two real cases.

An overview of the usability evaluation and its methodological foundation used to evaluate the prototype implementation and the method behind it are provided. Since the proposed method is user-centred, the method and the prototype are evaluated on a range of perception-based variables, namely, perceived ease of use, perceived usefulness and intention to use.

The chapter is structured as follows. In section 8.1, the aspects of evaluation are discussed, proceeding with a discussion of evaluation alternatives and restrictions in section 8.2. In section 8.3, the scope of evaluation is defined and the choice of evaluation method is described. In section 8.4, the design of the evaluation is described, followed by a presentation of the results and an analysis of the results in section 8.5. Finally, we revisit the requirements identified in chapter 2 and discuss how well they are satisfied by the method and implemented prototype in section 8.6. Section 8.7 summarises the chapter.

8.1 Evaluation Aspects

"Evaluation is concerned with gathering data about the usability of a design or product by a specified group of users for a particular

[14] who proves too much, proves nothing.

activity within a specified environment or work context.” (Preece et al., 1994, p. 602)

8.1.1 Qualitative and quantitative data

Usability evaluation is to gather data about the usability of a product. The results of an evaluation are used to improve a product or to compare it with existing products. In order for the evaluation to serve its purpose, the data should be appropriate. The collected data can be of quantitative or qualitative kind. Quantitative data (typically numbers) is used for statistical analysis describing how well a method performs. Qualitative data (typically text) is used to collect suggestions for improvements and document perceptions of test subjects.

Quantitative data are usually objective measurements, for instance, number of errors made, time spent on a given task and degree of required efforts. Alternatively, they can be subjective opinion about a specific feature of a product quantified into numbers. For instance, asking a subject to rate a feature using the Likert scale (Likert, 1931) from 1 (very difficult) to 5 (very easy).

Qualitative data is about understanding how and why. For instance, understanding what the user thinks, understanding why something is incomprehensible. This understanding usually is obtained either by observing the actual use of a product or allowing test subjects to specify themselves. Further, qualitative evaluations being concerned about how and why something works (or not), they can be useful during software development, for instance, a qualitative evaluation of design saves many hours of work.

Using quantitative data gives an advantage to process large amounts of results. Mean and standard deviation provides at-a-glance information regardless of the size of the collected data amount. However, using the quantitative data solely has a drawback. By quantifying a test subject’s opinion into numerical values, it is possible to lose quite a bit of detail. After all, knowing that a test group on average evaluates a specific feature (product) by a grade three out of five, provides no hint as to what works and what does not, and especially, why it does not work.

Because of the need to conduct an observation session or an interview, gathering qualitative data is more labour intensive than gathering quantitative data. As a result, fewer test subjects are involved.

8.1.2 Importance of test subjects

In order to get creditable and reliable results of an experiment right test subjects should be chosen. Scientists have long been aware that the answer to any given question depends on how many and who have been asked. In order to get a valid result, selection of right test subjects is important. For instance, it makes no sense to ask novices to evaluate a product intended for professionals – they are simply not a group representative for the intended audience.

For the creditability of the results, the amount of test subjects is important as well. Studies conducted by Nielsen (1993) have shown that as few as five users suffice in order discover a majority of usability problems. Increasing the number of testers beyond five was shown to generate marginal improvements. While a study conducted by Faulkner (2003), where random sets of 5 or more were sampled from 60 users, demonstrate the risks of using only 5 participants. As some randomly selected groups of 5 participants found 99% of the problems; other groups found only 55%. With 10 users, the lowest percentage of problems revealed by any of the sets was increased to 80%, and with 20 users, to 95%. Obviously, there is a risk when performing an evaluation with a limited amount of subjects. A lot depends on the subjects' qualification and motivation.

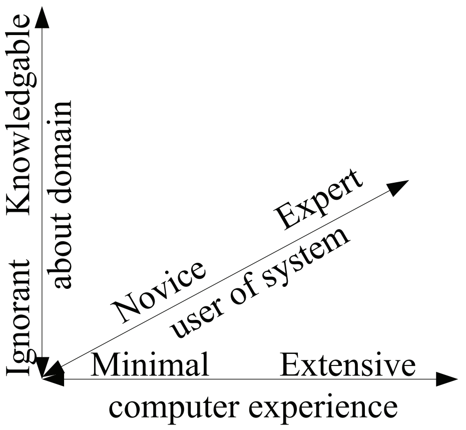


Figure 8.1 Classification of users (adapted from Nielsen (1993))

Therefore, Nielsen (1993) suggests classifying users according to three different axes regarding their computer experience, knowledge about the domain in question and experience with the system being evaluated. The three axes are illustrated in Figure 8.1. Locating a test subject in this three-dimensional space is usually done by means of background knowledge questionnaires.

8.2 Evaluation Alternatives

Recall, that in the previous chapters we have specified a method for product fragments management and change impact assessment using a conceptual domain model. The method is to be used in systems development, in particular, in distributed development. A part of the method concerns collaborative modelling as a means to achieve complete, shared understanding of a problem domain. This leads to various possible evaluation scenarios, each of them having restrictions and drawbacks when it comes to available resources allocated for this purpose.

Early design has been evaluated in a review group meetings consisting of 3-5 stakeholders and documented in (Strasunskas et al., 2004). Here we discuss alternatives to evaluate the implemented prototype.

As the proposed method targets distributed development, it is important to test it in similar settings, e.g., plenty of stakeholders, distributed environment, etc. One of the options is an industrial case study, either having the method adopted by a company, or replicating a real project. Adoption by a company is not feasible because of the immaturity of the methods itself and limited resources, e.g., financial – literally paying the company for using the method, and time – waiting for the results.

Enrolling students for similar size and scope project is more reasonable and also attainable within one semester. Getting them to participate in the experiment of such kind requires either running the experiment as a part of a particular course and giving them credit as for an assignment or paying for time spent for the experiment. The former way demands a prototype of good quality, especially when the experiment is part of the course assignment. The financial costs, required to pay students are lower if compared to an evaluation run by a company, are still too big.

Therefore, delimitation of the experiment scope is the only option in our case. That allows testing only a part of the method and a subset of the prototype functionality. Having the delimited scope, feasible directions for evaluation are as follows.

- *Testing collaboration and cooperation support.* Though, robust collaboration support is vital for distributed development, it is neither a primary concern, nor a contributing area for this thesis. There are specific application targeting a CSCW domain, and functionality supporting cooperative work of the most is much better, though different from ours. For that purpose a functioning CSCW environment would be needed. Since the scope of this study is only a small fraction of CSCW process, the results may be affected by the chosen "full scale" environment.
- *Testing support for externalisation and internalisation of knowledge* (i.e., in the modelling environment). Agreement on conceptualisation and full scalecase study does not lend itself to be measured because of time and financial limitations, as well as restricted implementation. Overall, in any smaller scope case – it is difficult to test to what degree agreement is possible. In a small group present at the same time and place it is possible to achieve common agreement. Though it would be interesting how the technology help to agree in distributed settings. Evaluation of this aspect is of high concern, even though because of limitations of the implementation (see chapter 7) the results could not be reliable.
- *Testing efficiency, effectiveness and usefulness of domain model based change management.* Evaluation of effectiveness may be conducted involving a

limited amount of test subjects. In our case, their task would be associating product fragments with domain model. The results from a log can be compared to an expert opinion about the dependency among product fragments or to the actual traceability links (if conducting a post-mortem analysis of a real project). Similarly, other de facto standard tools can be used instead of an expert. There subjects' efforts and results are analysed after performing the given task using both, the prototype and comparative tools.

- *Comparing prototype and state-of-the-art tools against a predefined set of requirements*. This evaluation scenario is simplest and cheapest to perform, though a bit biased, as the set of requirements for "must-be" functionality is difficult to prove being exhaustive. Moreover, requirements interpretation and especially interpretation of how well a particular functionality satisfies the requirements, is to high degree subjective. Furthermore, the implemented method enables different level of evaluation than rather comparing only how well the method satisfies the requirements listed in section 2.5.

Recall, that common conceptualisation of the domain and commitment to the domain model representing the conceptualisation is a cornerstone of our method. Though being important, it is not essential for the method to be accepted and adopted by the users. Most crucial for the method adoption is its efficiency and effectiveness of classification (association) of product fragments using a domain model when compared to the direct linking of related (dependent) product fragments. Therefore, we choose to validate this part of the method, i.e. domain model based change management. In addition, this part of the proposed method has the most robust implementation.

An experiment should seek evidence of efficacy of the proposed method as well as should gather and analyse users' opinion about the method and assess likelihood of its acceptance in practice. We devote section 8.4 to further elaborate the evaluation settings and design of the experiment. Next, an evaluation framework is discussed.

8.3 Evaluation Framework

In order to provide useful results an experiment should be systematically performed. Empirical study provides a means to evaluate the efficacy (efficiency and effectiveness), while feasibility and acceptance of the method are determined by measuring users' perceptions (Riemenschneider et al., 2002). Since we wish to measure performance of our method and user perceptions, we adopt the Method Evaluation Model (MEM) by Moody (2001), a model for evaluating IS design methods. The MEM incorporates both aspects as illustrated in Figure 8.2. Core of the MEM consists of the same perception based constructs as the Technology Acceptance Model (TAM) by Davis (1989), a model for explaining and predicting user acceptance of information technology. The constructs of the MEM are defined as follows.

- *Actual Efficiency*: the degree to which the method reduces the effort required to apply it;
- *Actual Effectiveness*: the degree to which the method improves the quality of the result;
- *Actual Usage*: the degree to which the method is used in practice;
- *Perceived Ease of Use*: the degree to which a person believes that using the method would be effortless;
- *Perceived Usefulness*: the degree to which a person believes that the method would be useful;
- *Intention to Use*: the degree to which a person intends to use the method.

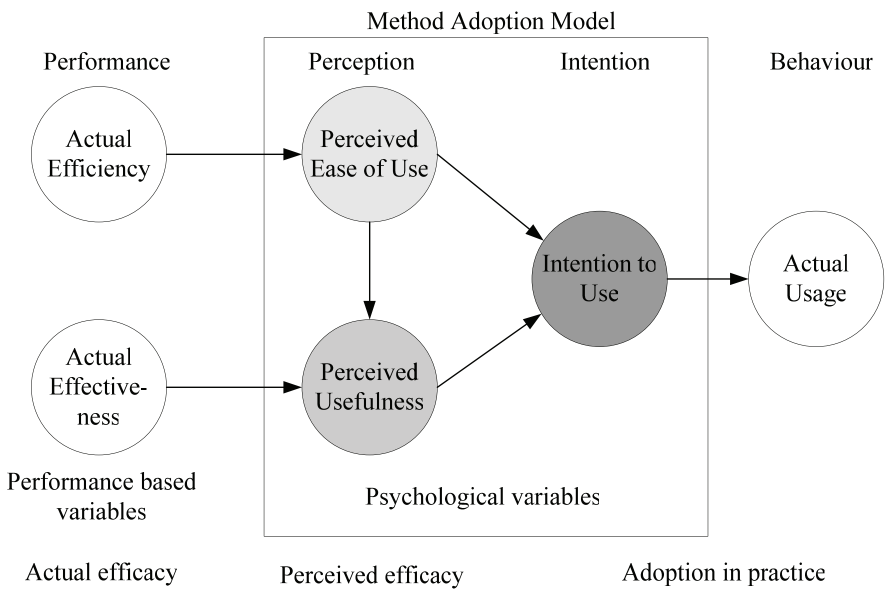


Figure 8.2 The Method Evaluation Model (adopted from Moody (2001))

Actual Efficacy measures whether the method actually improves task, while Perceived Efficacy represents perceptions of the method's efficiency and effectiveness. Adoption in practice is determined by perceptions, which are in turn determined by performance (Moody, 2001). Psychological variables are central constructs and are called the Method Adoption Model (MAM).

8.4 Organisation of the Experiment

User centred measurements are subjective and hence difficult to measure. For instance, one user may consider development objects being related (dependent), while another may find them totally independent. Our method to manage relatedness of the product fragments is user centric in the sense that semantics of the product fragments are best known to the creator of a particular fragment. The creator is the one who can best describe its internal semantics using the conceptual domain model in construction if which she has participated herself. Consequently, we organise the experiment in a way that best records users' perceptions.

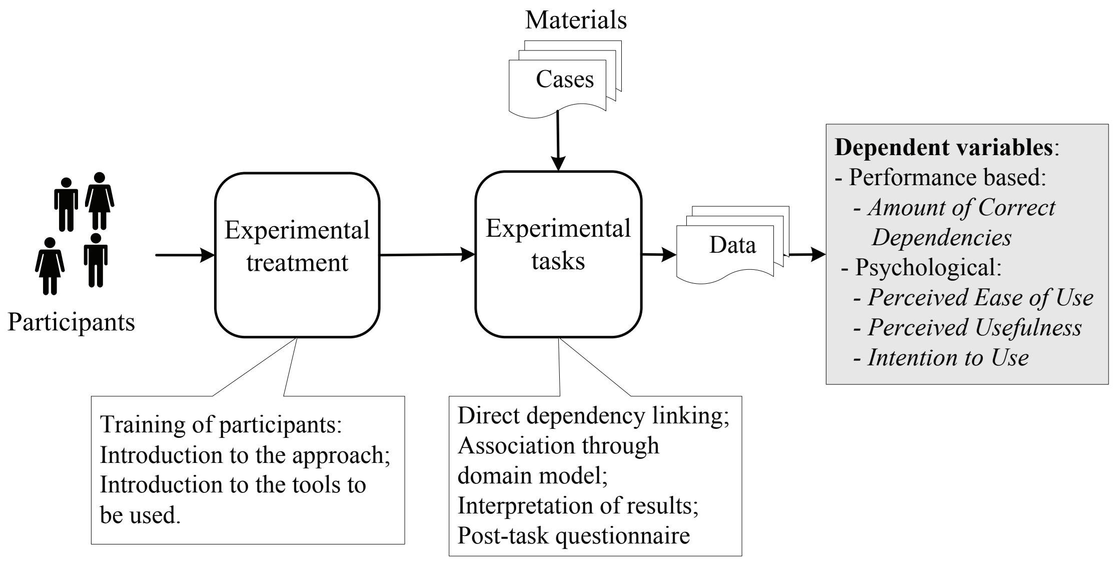

Materials
Cases
Participants
Experimental treatment
Experimental tasks
Data
Dependent variables:
- Performance based:
- Amount of Correct Dependencies
- Psychological:
- Perceived Ease of Use
- Perceived Usefulness
- Intention to Use
Training of participants: Introduction to the approach; Introduction to the tools to be used.
Direct dependency linking; Association through domain model; Interpretation of results; Post-task questionnaire

Figure 8.3 Design of the experiment

The experiment is designed to evaluate the effectiveness and usability of the proposed method for change impact assessment in a distributed development. The experiment design is summarised in Figure 8.3, where experimental treatment, experimental tasks, materials and collected data are shown.

This section further is organised according to according Figure 8.3 as follows. First subsection elaborates on rationale, states the goal of the experiment and evaluation questions. Second subsection decomposes the broad evaluation questions to hypotheses. Third subsection discusses the participant of the experiment and experimental treatment. Fourth subsection elucidates in detail the experimental materials, tasks given to the participants, etc. Fifth subsection discusses the dependent variables.

8.4.1 Rationale and goal of evaluation

In section 8.2 we have argued about the possible evaluation scenarios and restrictions concerning them. We have chosen to measure an actual effectiveness and users' perceptions about a domain model based change impact assessment.

Recall that the basis in our method is association of product fragments with a concept from a domain model, i.e. explicating semantics of product fragments. This association we interpret as a just another way of specifying metadata about objects, i.e. by relating the development objects to the structure of the problem, or in other words, classifying objects according the domain structure[15]. Metadata specification is known as labour intensive work and is not often used in practice, For instance, how many of us do describe metadata of documents we create, even if means are provided, for instance, using MS Word™[16] document properties. Therefore, we have chosen to test users' perceptions regarding ease of use and intention to use. Usefulness of the proposed method is tested analysing actual effectiveness and perceived usefulness. In addition, we want to collect the responses for further possible improvement of the method and prototype.

The goal of evaluation is to analyse and validate the proposed method implemented prototype regarding its effectiveness and likelihood of adoption in practice from the point of view of possible users. The broad evaluation questions (EQs) addressed by this evaluation are as follows.

- **EQ1**: Is the method effective?
- **EQ2**: Is the method apt to be adopted in practice?

8.4.2 Hypotheses

A priori, the assumed effects of using the domain model to manage relatedness of product fragments and theirs change impact are hypothesised as follows.

- **H1**: The method is effective, i.e. domain model facilitates dependency establishment among product fragments and helps to explore relatedness of product fragments and discover "hidden" dependencies,
- **H2**: The method is perceived as easy to use,
- **H3**: The method is perceived as useful,

and consequently,

- **H4**: There is an intention to use the method.

8.4.3 Participants selection and experimental treatment

Since the proposed method is meant to be used by a variety of stakeholders, we've selected test subjects with different background, though all of them from computer science area (information management, databases, information systems, knowledge management). All six test subjects are from Dept. of Computer and Information Science at NTNU.

The subjects received 45 minutes long training session. First, the method to be tested and overall idea was presented. Then, an evaluation task and procedure was presented, demonstrated and discussed at a common meeting. Finally, short tutorials on each of the tools to be used in the experiment were given.

[15] Similarly to saving a file into the designated folder on file system.

[16] Microsoft® Word™ is a registered trademark of Microsoft Inc.

Couple of subjects were familiar with the concepts of the method and have seen the prototype, but had not been using. Other subjects had not seen the tool before. Using user classification axes provided by Nielsen (1993) and discussed in section 8.1, the test subjects have extensive computer experience (5 have 11-25 years experience, and only one 5-10 years experience). Furthermore, 5 subjects have an industrial experience. None of the test subjects is an expert user of the tested tools (see next subsection), in fact only one had experience of using traceability matrix before. None of the users had addressed the problem of dependency management prior to the experiment, with the exception of one, whom had been working earlier with dependencies and traceability links.

Subjects were volunteers. Since no resources were available for rewarding or paying users for this evaluation, the only reward given for participation was a free private dinner[17]. Furthermore, four out of six experiment participants were PhD students, for them additional motivation was to learn from evaluation process itself as they will need to perform evaluation in a short future themselves, as well as they will need to get volunteers for testing their tools/methods, i.e. their motivation was driven by *quid pro quo*[18] principle.

8.4.4 Experimental materials

The instrumentation used in the experiment included experimental materials, tools (the studied prototype, Telelogic® Doors™ (Telelogic, 2005), and traceability matrix implemented in MS Excel™), a log for performance measurement and survey techniques (questionnaire and observation with "think aloud" protocol).

The experimental materials consisted of two cases and their descriptions in natural language, domain models in RML (Appendix A), and diverse product fragments (Appendix D). Case 1 was taken from MSc project (Erichsen, 2003). The MSc project was similar in to the method proposed here, i.e. dealing with dependency and traceability. Case 1 consisted of 4 different product fragment types, i.e., requirement statements (natural language), design fragments (UML™ sequence diagrams), code fragments (C#) and user manual in a form of screenshots. Case 2 was based on the development materials of MEIS (Model Evaluation Information System) system, used for the introductory course on information systems TDT4175 (Matulevicius et al., 2004). MEIS system is used for exercise delivery, peer-to-peer review and evaluation of both. Case 2 had two types of product fragments, i.e. requirements statements (natural language) and code (php).

Description and a domain model for Case 2 are exemplified in Example 8.1 below. For more details see appendix D, there is provided the list of product fragments for each of the cases and typical product fragments of each type are

[17] Food in Norway is regarded as a quite good means to increase motivation.
[18] In Latin, mutual consideration; service in return.

illustrated. A list of the product fragments for Case 2 is provided in Table 8.1, this table is replicated here from Appendix D in order to exemplify the scope.

Conceptual domain models that the researcher designed had 18 concepts 21 relationships in case 1 and 18 concepts 26 relations in case 2. Case 1 had 83 product fragments, and case 2 had 23 fragments. For both cases, the task assigned to the subjects was to study the materials and afterwards, specify dependency relationships for a set of selected development objects. The tasks are specified below.

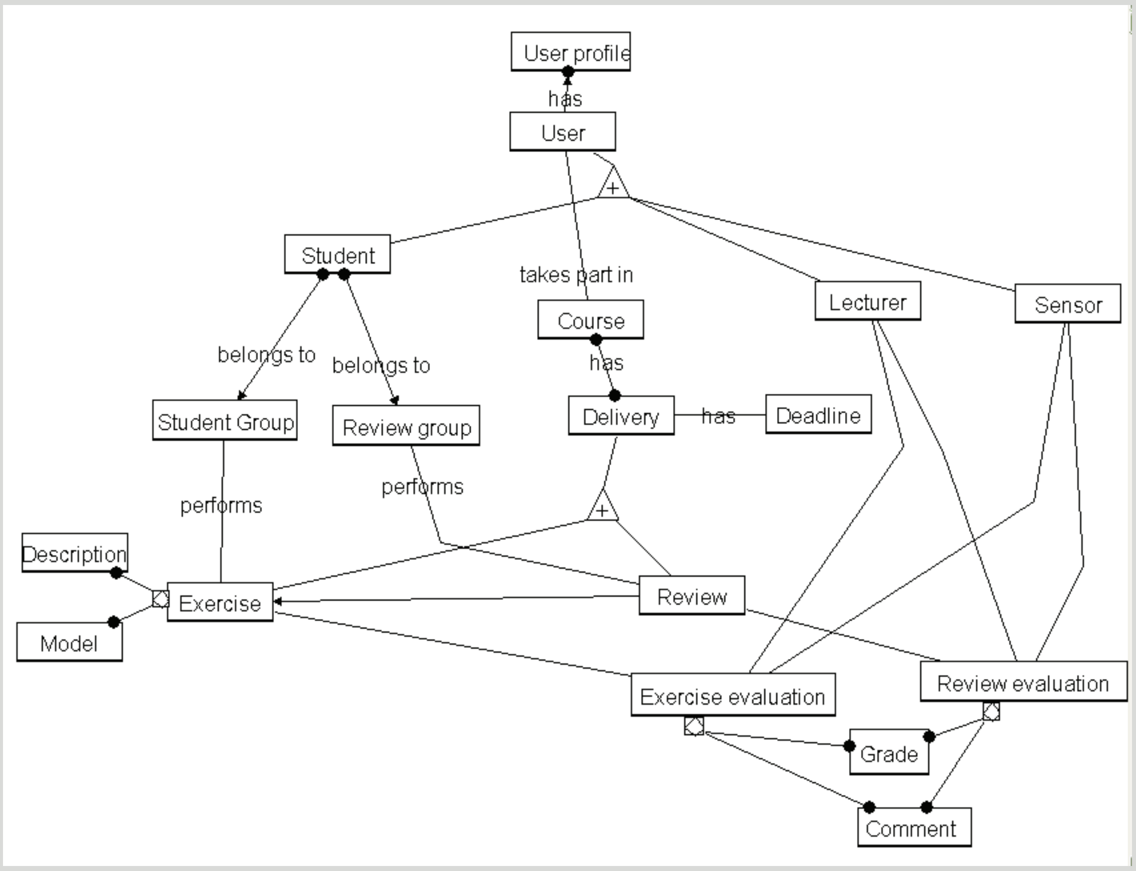
Example 8.1 Case 2 description and domain model

This case describes a project developing a system at NTNU to be used to support exercise delivery, review and evaluation during for a particular courses. A simplified model for Case 2 is depicted in figure 1 for the purpose of defining the scope.

User is involved (takes part) in the *course*. There are three types of users, namely, *student*, *lecturer*, and *sensor*. Every user of the system has a *user profile*. There are two types of *delivery* in the course, i.e., *exercise* and *review*. Each delivery has a *deadline*.

Students are organized in *student groups* to deliver an exercise. Solution to an exercise usually consists of a *description* and a *model*. After delivering the exercise students are arranged into *review groups*, in order to peer-review delivered exercises.

Lecturer and sensor perform *evaluation* of both exercise and review of exercise, by assigning a *grade* and providing some feedback (*comment*).



Conceptual domain model for case

There are two different types of development objects in this case. Namely, requirement statements, and code (php) fragments.

Your task now is to:

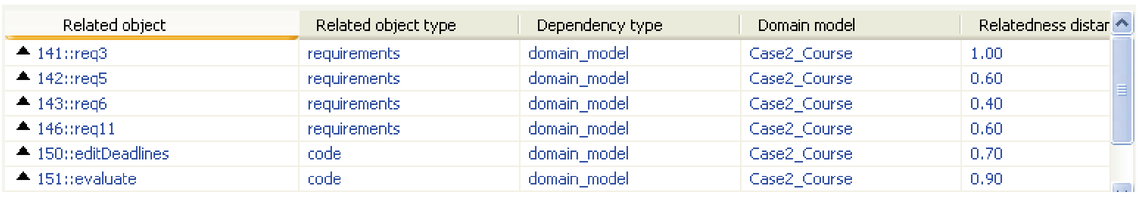
- Study the materials
- Specify relationships for selected (marked) development objects

Table 8.1 List of the product fragments in case 2

Product fragment		
Type	**Name**	**Description**
Code	Deliver	control of delivery form
	Deliveries	overview of deliveries
	doEvaluate	control of evaluation form
	Evaluate	control of reviewing form
	Feedback	control of feedback form
	index	start form (login)
	updateReview	form for changing the review
	updateUserinfo	form for user profile update
	upload	exercise upload form and control
	userinfo	form for browinsg user profile
	viewDeliveryComments	form for browsing delivery evaluation (from student view)
	viewEvaluation	form for browsing evaluation of delivery
	viewFeedback	form for viewing received feedback (from lecturer view)
	viewReviewComments	form for browsing received review comments (from student view)
	editDeadline	form for setting and changing the deadlines
	viewGroups	set and browse students arranged into groups
Requirements	req3	Student should be able to log in to the system
	req5	Student should be able upload delivery (exercise)
	req6	Lecturer should be able to see the status of the deliveries from the assigned students.
	req8	Student should be able see the comments for solution
	req9	Lecturer should form a reviewer groups for delivery
	req11	Reviewer should be able evaluate the deliveries

Experimental task

First, the test subjects needed to study the experimental materials. *Second*, the subjects were asked to do both, establish the direct dependency links among fragments (see Table 8.1 above) and associate the fragments with concepts from the provided domain model (see Example 8.1 and Table 8.1 above). *Third*, after running adjusted algorithm (see discussion below) for change impact assessment based on the provided domain model, they needed to choose three random product fragments (see again tables D.1 and D.2 in Appendix D for the list of product fragments) and investigate three top ranked relatedness values (not items, meaning that amount investigated items may differ for each of the subjects) provided by the studied prototype. Figure 8.4 presents a screenshot to exemplify the results set provided by our prototype. In the figure are listed product fragments related to the product fragment "deliver" (for details see specification of Case 2 in Appendix D). The last column in Figure 8.4 shows relatedness value. Recall discussion in chapter 6 that the relatedness value here is calculated based on shortest path between two particular fragments, i.e. the smaller value shows that the fragments are more likely dependent. The subjects were asked to treat this list as indication that a pair of product fragments is related and might be dependent. *Fourth*, they needed to investigate the list and classify the fragments returned by the prototype as *totally wrong* (having nothing to do with each other), *partially correct* (for those that seems to be dependent, but dependency need more detail investigation), or *totally correct* (the fragments are dependent).



Related object	Related object type	Dependency type	Domain model	Relatedness distar
141::req3	requirements	domain_model	Case2_Course	1.00
142::req5	requirements	domain_model	Case2_Course	0.60
143::req6	requirements	domain_model	Case2_Course	0.40
146::req11	requirements	domain_model	Case2_Course	0.60
150::editDeadlines	code	domain_model	Case2_Course	0.70
151::evaluate	code	domain_model	Case2_Course	0.90

Figure 8.4 Example of computed dependency for the fragment "deliver" from case 2

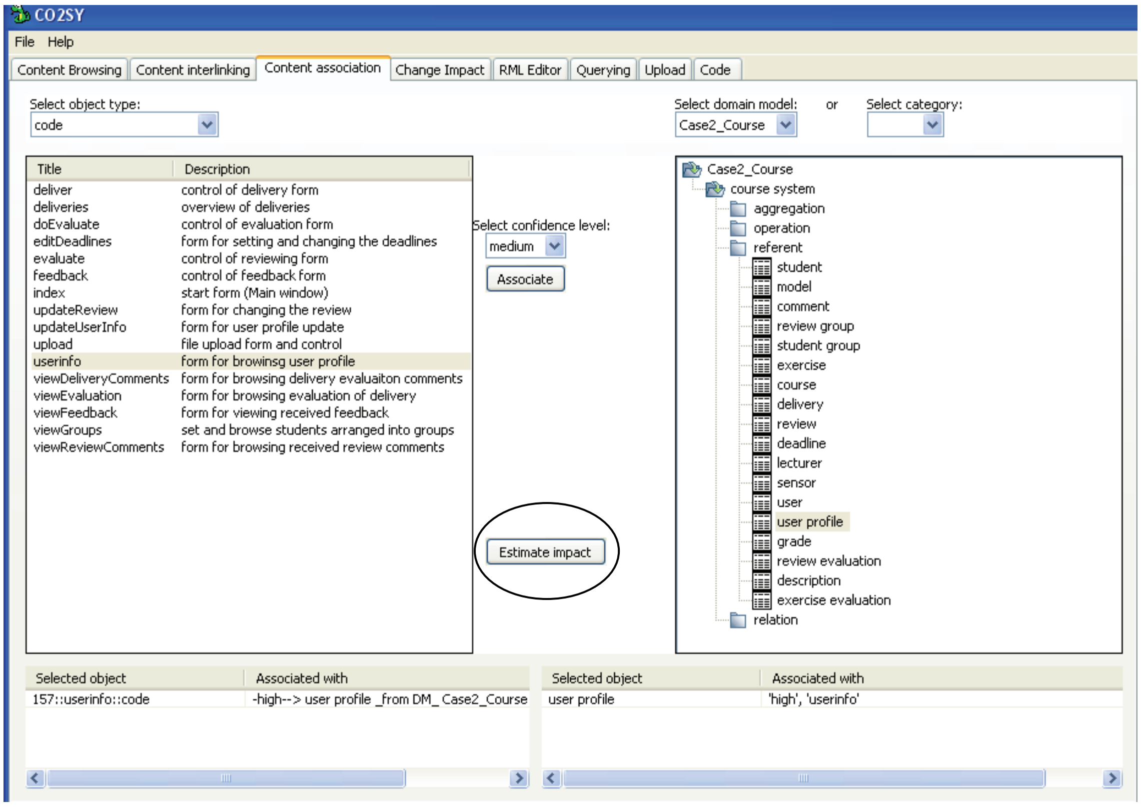
Later, the pairs identified as *totally correct* were compared to the set of directly linked product fragments for each user. Consequently, the set of totally correct dependency links identified by the prototype was divided into two subsets, namely, *mutual* (i.e. identified by the user when performing the task of direct linking) and *additional* (i.e. proposed by the prototype, but did not noticed by the user). These results are discussed in section 8.5 and presented in Table 8.2.

Comparative tools

In addition to the prototype, two tools were used, namely, Telelogic® Doors™ and a traceability matrix implemented in MS Excel™. Both consider direct linking of related product fragments, but by the means of different interfaces. Thus, the experiment would not be biased by interface. The test subjects were in two groups. Group A first established direct links using Doors (case 1) and traceability matrix (case 2), then associated product fragments using the prototype. While group B first assigned associations using the prototype (with materials of both cases), and then direct linking using the comparative tools.

Prototype set-up and modification

Prototype needed to be modification according to the settings of the experiment. Recall the intentional use of the method discussed in chapter 6. After domain has been successfully conceptualised, it is used to classify the product fragments, i.e. by associating product fragments with concepts. Every time new product fragment is produced, it is associated with one or more concepts. In this way the process of association is not labour intensive. Then, whenever new revision is checked-in to the repository, the algorithm for change impact assessment is triggered, i.e. notifications are fired. While in the experiment, test subjects were asked to associate a set of product fragments with corresponding concepts, after having done that, the impact assessment algorithm is triggered. For that reason additional button needed to be created (see Figure 8.5) to start algorithm.



Select object type: code

Select domain model: Case2_Course or Select category:

Title	Description
deliver	control of delivery form
deliveries	overview of deliveries
doEvaluate	control of evaluation form
editDeadlines	form for setting and changing the deadlines
evaluate	control of reviewing form
feedback	control of feedback form
index	start form (Main window)
updateReview	form for changing the review
updateUserInfo	form for user profile update
upload	file upload form and control
userinfo	form for browinsg user profile
viewDeliveryComments	form for browsing delivery evaluaiton comments
viewEvaluation	form for browsing evaluation of delivery
viewFeedback	form for viewing received feedback
viewGroups	set and browse students arranged into groups
viewReviewComments	form for browsing received review comments

Select confidence level: medium

- Case2_Course
  - course system
    - aggregation
    - operation
    - referent
      - student
      - model
      - comment
      - review group
      - student group
      - exercise
      - course
      - delivery
      - review
      - deadline
      - lecturer
      - sensor
      - user
      - user profile
      - grade
      - review evaluation
      - description
      - exercise evaluation
    - relation

Selected object	Associated with
157::userinfo::code	-high--> user profile _from DM_ Case2_Course

Selected object	Associated with
user profile	'high', 'userinfo'

Figure 8.5 Modified window for association with concepts

Survey

At the end of the evaluation, the test subjects answered a questionnaire (see Appendix C) in order to obtain general feedback of the system as well as to discuss the drawbacks. To gain deeper insight into subjects' views, "think aloud" protocol was used, where subjects were asked to think aloud while performing the tasks. All subjects have been surveyed by means of the questionnaire. Most of the questions were closed, but open questions were also included to allow for unanticipated reactions. The questionnaire included 21 closed questions, 3 closed questions with unordered responses, and 3 open questions. A five-point Likert scale (Likert, 1931) is used to measure 13 out of 21 closed questions, i.e. subjects are asked to express agreement or disagreement of a five-point scale. Each degree of agreement is given a numerical value from one to five. Thus a total numerical value can be calculated from all these responses.

8.4.5 Dependent variables

We distinguish two types of dependent variables (recall Figure 8.3): performance based and psychological variables. Evaluation of the actual efficacy requires measuring the efforts needed to use method and the resulting quality. To evaluate the actual effectiveness we have chosen to measure one performance based variable.

- *Amount of correct dependencies* is measured as comparison of results for subject from direct dependency linking and associating through domain model. This construct was analysed using a log and one question (i.e. earlier mentioned practical task, question 6) from the post-task questionnaire.

To evaluate the perceived efficacy and intention to use we adopt the three psychological variables of the MAM. They are as follows.

- *Perceived Ease of Use* (PEU) measured using four questions[19], i.e. 13, 14, 15, and 16;
- *Perceived Usefulness* (PU) measured using two questions, i.e. 7 and 18;
- *Intention to Use* (IU) measured using three questions, i.e. 10, 11, and 24.

The order of the questions in the questionnaire was randomized to avoid monotonous responses. To avoid a possible ceiling effect, there was no time limit for the experiment to restrict subjects.

8.5 Results

This section presents the results from the performed evaluation. First, we present overall results, analyse the results using the above described method, namely, actual effectiveness, perceived ease of use and usefulness, intention to use. Second, we analyse subjects' behaviour and performance, and investigate what impact it possible has to the results. Third, we summarise responses to open-ended questions. Finally, we discuss threats to validity.

[19] See Appendix C for particular questions in the questionnaire.

8.5.1 Overall results

Time

In average, the subjects used 3,5 hours to perform the task. Approximately 3/5 of the time was used to study the experimental materials. There are three reasons why measurement of time is not used to calculate efficiency. Namely, 1) the subjects were not studying the material equally before and during the actual linking; 2) some subjects found "short cuts" in some interfaces than the others, for instance in Doors package it is possible to make the link using formal link module, then the process goes through several dialog boxes, or more efficiently is a simple "drag and drop"; and 3) some encountered the lack of "undo" command in the studied prototype more than others. An interaction with the tools is otherwise assumed to take similar time if disregarding the routines forced by the interface. Analysis of the log, showed that in average a fragment had 3,8 (in case 1) and 2,5 (in case 2) directly linked fragments. Similarly, a fragment had been associated with 3,3 (in case 1) and 3,1 (in case 2) concepts in average (see Table 8.10). These parameters are similar, i.e. the time used is comparable. In retrospect, overuse of the time (i.e. for first and third reasons listed above) could have been recorded and then deducted from the time spent using a particular tool.

Actual efficacy, i.e. amount of correct dependencies

Recall, that the subjects were asked to classify the correctness of the output. The results for our method are displayed in Table 8.2. Where all partially correct are new dependency pairs for the users, i.e. additional links discovered by the prototype. Correctly identified dependency pairs are grouped exclusively as mutual (identified by the test subject as dependent already by direct links) or additional knowledge (new dependency links, proposed by the prototype and identified being correct by the test subject), see the columns 4 and 5 in the table.

Table 8.2 Analysis of dependency discovery performance

	Subject ID	Totally wrong	Partial	Totally correct			Total inspected	Total additional	% of correct	% of possbily correct (incl. Partial)	% of wrong
			additional	mutual	additional	%					
	(1)	(2)	(3)	(4)	(5)	(6) =(5)/((4)+(5))	(7) =(2)+(3)+(4)+(5)	(8) =(3)+(5)	(9) =((5)+(4))/(7)	(10) =((8)+(4))/(7)	(11) =(2)/(7)
Case 1	*1*	0	3	5	1	17 %	9	4	67 %	100 %	0 %
	2	4	3	4	0	0 %	11	3	36 %	64 %	36 %
	3	5	8	12	3	20 %	28	11	54 %	82 %	18 %
	4	2	0	13	2	13 %	17	2	88 %	88 %	12 %
	5	6	1	4	3	43 %	14	4	50 %	57 %	43 %
	6	6	1	5	0	0 %	12	1	42 %	50 %	50 %
	Total	**23**	**16**	**43**	**9**	**17 %**	**91**	**25**	**57 %**	**75 %**	**25 %**
Case 2	*1*	2	3	4	0	0 %	9	3	44 %	78 %	22 %
	2	3	2	3	0	0 %	8	2	38 %	63 %	38 %
	3	1	3	7	0	0 %	11	3	64 %	91 %	9 %
	4	5	3	4	1	20 %	13	4	38 %	62 %	38 %
	5	1	2	6	2	25 %	11	4	73 %	91 %	9 %
	6	9	1	5	0	0 %	15	1	33 %	40 %	60 %
	Total	**21**	**14**	**29**	**3**	**9 %**	**67**	**17**	**48 %**	**69 %**	**31 %**

In case 1 the prototype has discovered 9 additional correct links (column 5), and 16 possibly correct (column 3). Overall, 17% (column 6) of correct dependency links identified by the prototype was a new knowledge for the test subjects, i.e. the dependencies overlooked when making direct dependency links. While the amount of total additional (both, totally correct and partially correct) equals to 25 (column 8). In case 2, the results are slightly worse, i.e. only 3 dependency pairs were considered being correct additions to the set already identified by the users. That makes 9% being new knowledge to the test subjects. Total additional dependency links identified by the prototype sum up to 17 (column 8). Effectiveness of the method is proved by increased recall compared to manually linked product fragments. Therefore, hypothesis $\mathbf{H_1}$ is confirmed. However, section 5 presents more detailed analysis of the difference between the results with case 1 and case 2.

Perceived Ease of Use

Cramer (1994) argues that ordinal scales are often treated as interval because researches pay less attention to the levels of measurement than is paid to the statistical test of choice. Cramer (1994) notes that parametric statistics rely on the estimated population 'parameter' from a sample, and this usually makes three assumptions about the data. First, the level of measurement is non-categorical (or interval/ratio quality). Second, the variances of any comparison between different groups of such data are equal. Third, the data is distributed normally. Cramer (1994) notes that research shows that violation of either of the latter two assumptions do not impact on results, but violation of both is problematic. Therefore, for this construct, we first analyse the normality of answers distribution, and then we proceed to parametric test of data.

Hypothesis H_2 can be statistically testes by verifying whether the scores that subjects have given to the questions related to the constructs of MEM are significantly better than the middle score, i.e. the score 3 on the Likert scale for the question. The score 3 means, that a subject's perception is neutral, i.e. the method was not perceived neither easy nor difficult to use. If subject's rating is higher than the middle score, then he/she perceives an advantage of the method. Therefore, the null hypothesis for the hypothesis H_2 is formulated as follows.

- $\mathbf{H_{2N}}$: The perception of the method being ease of use is neutral.

Figure 8.4 shows an average score for each of the subjects, calculated from the responses to the PEU relevant questions. The One-Sample Kolmogorov-Smirnov test (with Normal theoretical distribution) is applied to the answers related to the constructs of the PEU (see Table 8.3). The distributions is normal, i.e. *p* values are high (lowest was 0,33 for Q13 Ease of Using the prototype) i.e. distribution is quite normal. Therefore, one-tailed t-test was used to check for the difference in mean of PEU construct and the middle score value 3. To evaluate the significance of the observed difference, we applied a statistical test with a significance level of 5%.

Table 8.3 One-Sample Kolmogorov-Smirnov Test for responses measuring PEU

		Q13_EoU	Q13_EoL	Q14	Q15	Q16
N		6	6	6	6	6
Poisson Parameter	Mean	3,17	3,83	4,17	3,67	3,83
Most Extreme	Absolute	0,387	0,339	0,241	0,291	0,339
Differences	Positive	0,223	0,339	0,241	0,165	0,339
	Negative	-0,387	-0,300	-0,235	-0,291	-0,300
Kolmogorov-Smirnov Z		0,948	0,829	0,591	0,713	0,829
Asymp. Sig. (2-tailed)		0,330	0,497	0,875	0,689	0,497

Remark: EoU – Ease of Using; EoL – Ease of Learning.

Table 8.4 Mean scores assigned by test subject for each construct of the MEM

Subject ID	**1**	**2**	**3**	**4**	**5**	**6**
Perceived Ease of Use	4,00	3,20	3,40	4,20	3,60	4,40

Table 8.5 provides descriptive statistics for the PEU construct.

Table 8.5 Descriptive statistics for PEU.

Number of observations	**Minimum**	**Maximum**	**Mean**	**Std. Deviation**	**Std. Error Mean**
6	3,200	4,400	3,800	0,473	0,193

The results in Table 8.6 allow for the rejection of the null hypothesis H_{2N}, meaning that we empirically corroborated that participants perceived the tool and method to be easy to use.

Table 8.6 One Sample t-test for difference in mean

t	**1-tailed p**	**Mean difference**	**95% Confidence Interval of the difference**	
4,140	0,005	0,800	0,303 (lower)	1,297 (upper)

Next we measure the reliability of PEU construct. The reliability measure describes consistency the construct gives in measuring the same phenomenon over time or by different people. Cronbach's alpha for the construct PEU is 0,86 (usually values over 0,7 are expected in order for construct to be reliable).

Table 8.7 Total item statistics

Questionnaire item related to the PEU construct	**Scale Mean if Item Deleted**	**Scale Variance if Item Deleted**	**Corrected Item-Total Correlation**	**Cronbach's Alpha if Item Deleted**
Q13_EoU_prototype	32,67	35,867	-,382	,889
Q13_EoL_prototype	32,00	31,600	,523	,857
Q16_prototype	32,00	33,600	,085	,873
Q15_prototype	32,17	32,167	,144	,879
Q14_prototype	31,67	26,267	,950	,819
Q13_EoU_doors	32,17	23,367	,797	,825
Q13_EoL_doors	32,67	27,467	,777	,833
Q16_doors	32,17	22,167	,924	,809
Q15_doors	32,67	27,467	,777	,833
Q14_doors	32,33	21,067	,885	,815

Table 8.7 shows that all items are consistent, i.e. Cronbach's alpha is still above 0,8 if any of the items deleted. So, we conclude that the items used to measure perceived ease of use are reliable and valid measures for this perception based construct.

Perceived Usefulness

Recall, that the perceived usefulness we have measure using two items from the questionnaire, namely, question 7 and question 18. As answers to question 7 were identical, i.e. all test subjects answered that the prototype helped to discover *some* new correct dependency links. While answering to the question 18, two test subjects have identified results of the prototype being accurate as neutral, i.e. middle value in a scale from *1=Total disaster* to *5=Very accurate*. Others have chosen value 4. Because of small number of questions covering this metric, we are not able to apply the same calculation as for PEU construct. Cronbach's alpha is not computable using SPSS, as one variable is constant, i.e. all subjects have given the same answer to question 7 (see Appendix E).

Table 8.8 Comparison of responses to Q6 and Q18

Subject ID	**1**	**2**	**3**	***4***	**5**	***6***
# of total inspected	18	19	39	*30*	25	*27*
# of total additional	7	5	14	*6*	8	*2*
% of additional	38,9%	26,3%	35,9%	*20,0%*	32,0%	*7,4%*
Q18	4	4	4	*3*	4	*3*

In order to validate users' consistency in answering, we have compared answers to question 18 with actual effectiveness, i.e. answers to the question 6. Data in Table 8.8 are taken from Table 8.2. From Table 8.8 it is obvious that both users selected value 3 in question 18 had least percentage of additional correct dependency links discovered by the prototype. So, we can treat their answers being honest and consistent.

To summarise, PU construct shows that the prototype usefulness has positive perceptions among users. In section 8.5.2 we return to the analysis why 1/3 of users have ranked accuracy of prototype results as "neutral" (average).

Intention to Use

Intention to use was measured by three items in questionnaire, i.e. questions 10, 11 and 24. Users were asked to rank the tools they have used in preferable to use order, i.e. assigning 1st, 2nd or 3rd place. Because of specificity of the response format, we certainly cannot use any of parametric tests. Therefore, we have chosen to use Kendall coefficient of concordance W (Siegel & Castellan, 1988) to measure agreement among users' ranking.

Kendall coefficient of concordance W is computed according Eq. 8.1.

$$W = \frac{\sum_{i=1}^{N}(\overline{R_i} - \overline{R})^2}{N(N^2-1)/12} \qquad \textit{Eq. (8.1)}$$

Where,

N = number of object being ranked;

$\overline{R}$ = the average of the ranks assigned across all objects;

$\overline{R_i}$ = average of the ranks assigned to the i^{th} object.

Table 8.9 displays the ranks given by users and average of the ranks, the average of the ranks assigned across all objects is equal to 2,00. Kendall coefficient of concordance W_{Q10} = 0,58 for question 10 and W_{Q11} = 0,86 for question 11.

Table 8.9 Responses to Q10 and Q11 with an average of ranks

Question	**Q10**							**Q11**						
Subject ID	**1**	**2**	**3**	**4**	**5**	**6**	$\overline{R_i}$	**1**	**2**	**3**	**4**	**5**	**6**	$\overline{R_i}$
Prototype	1	1	1	2	1	1	*1,17*	1	1	1	1	1	1	*1,00*
Doors	3	2	2	1	2	3	*2,17*	3	2	2	2	2	2	*2,17*
Traceability matrix	2	3	3	3	3	2	*2,67*	2	3	3	3	3	3	*2,83*

When answering to question 24 to select what tool they would like to use it in future, all test subjects selected the prototype, though subject #2 in addition selected traceability matrix, and subject #4 – Doors tool. Subject #4 has provided justification that "Doors is more suitable for moderate-sized project", "prototype would be great for large scale / distributed development".

Based on the above presented data analysis, we can claim that there is an intention to use the tool (method), i.e. **H_4** is confirmed.

8.5.2 Discussion

Here we analyse the results and especially the difference in results in more detail. First, we analyse a possible cause; second, we discuss and relate the results to metrics used in dependency (traceability) links establishment, i.e. related approaches. We conclude the discussion by analysis of possible threats to validity and summary of observations with a feedback received from the test subjects.

Cause analysis

In order to investigate the difference between the results with case 1 and case 2 respectively, we have analysed an average amount of concepts associated with a product fragment (see Table 8.10). Generally, it can be observed that the bigger concept cluster the bigger result set is produced, i.e. the result set will contain more false positive. There are no significant correlations between an average concept cluster size (Table 8.10) and the result set (Table 8.2). Namely, correlation coefficient between the mean of cluster size and amount of wrong identification is 0,14 in case 1 and -0,52 in case 2. While correlation coefficients between the mean of cluster size and amount of total inspected identifications are -0,16 in case 1 and -0,04 in case 2.

However there is a notable difference in a cluster size between group A and group B. The difference exists in the both cases. This difference in an average amount of concepts used to describe semantics of a product fragment is a outcome of different usage sequence of experimental tools. Group B has started performing task with a CO$_2$SY prototype, and after proceeding to direct linking using the comparative tools. While group A did it other way around. Group A needed more thoroughly investigate the product fragments when performing direct dependency linking task, i.e. they have had more clear perception of semantics (content) of the product fragments when associating them with the concepts.

Table 8.10 Mean of concepts cluster size associated per fragment

Subject ID	Case 1					Case 2		
	Mean				Overall mean (± st.dev.)	Mean		Overall mean (± st.dev.)
	code	require-ments	user manual	design		code	require-ments	
(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)
1	2,3	3,0	2,0	2,3	2,4 (±0,8)	1,2	2,5	1,9 (±1,0)
2	3,0	4,5	1,5	4,0	3,3 (±1,4)	3,2	2,8	3,0 (±2,4)
3	2,1	2,2	1,9	3,0	2,3 (±1,2)	3,4	4,8	4,1 (±2,1)
4	4,5	5,5	4,5	4,0	4,6 (±0,7)	4,2	1,3	2,7 (±2,5)
5	4,5	4,5	4,0	5,0	4,5 (±1,1)	4,4	4,3	4,3 (±1,0)
6	4,0	4,5	1,5	2,0	3,0 (±1,5)	2,0	3,0	2,5 (±1,4)
Overall					**3,3 (±1,4)**			**3,1 (±2,0)**
group A					*2,3 (±1,2)*			*2,7 (±1,0)*
group B					*4,1 (±1,2)*			*3,4 (±2,1)*

Next, we have analysed the responses about the quality of case 1 and case 2. We assume that better quality of domain model and case description should facilitate association of product fragments with concepts, while worse quality of the product fragments makes direct linking more difficult. The results are summarised in Table 8.11 and graphically displayed in Figure 8.6. Values in Table 8.11 are displayed as count of answers, i.e. three users meant that quality of fragments in case 1 was fair. Weighted total quality is calculated using equation 8.2. Median is calculated based on responses using a five-point Likert scale, i.e. "very bad" = 1, "very good" = 5.

$$WT = \sum_{i=1}^{5} (w_i \times |V_i|) \qquad \textit{Eq. (8.2)}$$

Where, `VC` is a set of values categories, i.e. `VC = {very bad; bad; fair; good; very good}`, V_i is a set of all occurrences of the response type from `VC`, $v \in VC$ and $v \in V$; `W` is a set of weights, $w \in W = \{-2; -1; 0; 1; 2\}$.

The quality of case 2 description and domain model was perceived much better than of case 1 (in addition 50% of test subjects identified lack of domain knowledge in case 1, see answers to question 12 in Appendix E), while the variation of fragments' quality is not so big. Obviously, that does not explain the differences of results in Table 8.2. Small variation in perceived quality of product fragments suggests, that direct linking should have been easier in case 1, meaning *less additional correct links* identified by the prototype. However, one test subject noted that fragments were more related to the structure of the problem in case 1, whereas in case 2 fragments seemed to be related to the structure of program (software). That sounds reasonable and explains the results in Table 8.2, as case 1 was based on the implementation materials from MSc project, while case 2 was based on implementation materials from the system actually used.

Table 8.11 Perceived quality of cases

Quality of		**Very bad**	**Bad**	**Fair**	**Good**	**Very good**	**Weighted total**	**Median**
(1)	**(2)**	**(3)**	**(4)**	**(5)**	**(6)**	**(7)**	**(8)**	**(9)**
Case 1	description	-	1	4	1	-	**0**	3,0
	domain model	-	-	4	2	-	**2**	3,0
	fragments	-	1	3	1	1	**2**	3,0
Case 2	description	-	-	2	4	-	**4**	4,0
	domain model	-	-	1	5	-	**5**	4,0
	fragments	-	2	2	2	-	**0**	3,0

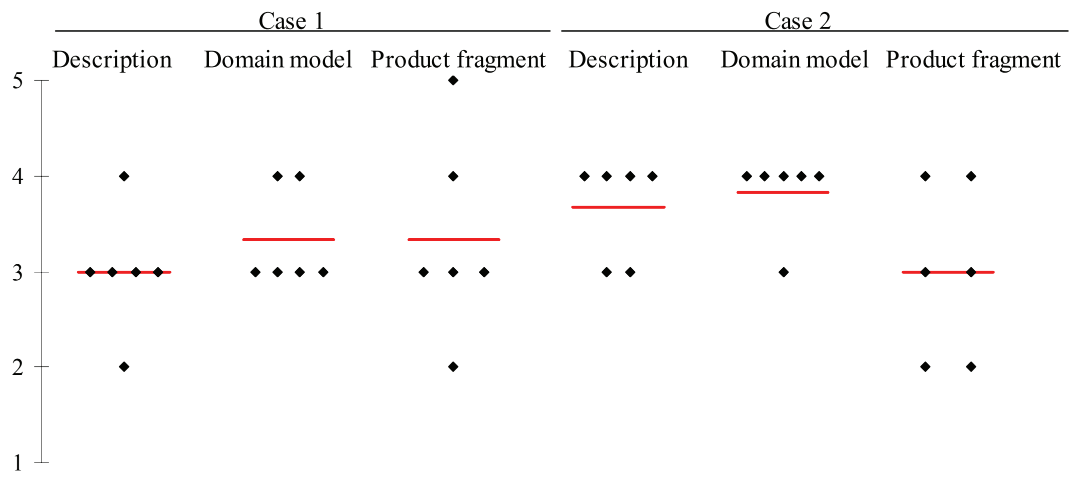


Figure 8.6 Perceived quality
Mean is denoted by line, dots represent chosen value by test subjects

Further, in order to analyse the variance of the subjects' perceptions regarding PU construct (recall the variance between the users 4 and 6 in the mean of concepts associated with product fragment (see Table 8.10)), we decided to take a look at users' pattern using confidence level when associating with concepts[20]. It is reasonable, since high confidence level of association gives a numerical value of 0,2 (0,5 is for medium and 0,8 is for low confidence level), i.e. as discussed in chapter 6. Table 8.12 shows that, actually user #4 used only high confidence level (100%) and user #6 used over 90% of high confidence level for associations. This was the reason to compute a lot of false positive impact notifications and obviously had influenced the reason for different answers for the construct of Perceived Usefulness.

Table 8.12 Percentage of different confidence levels used for associations

Subject ID	**1**	**2**	**3**	**4**	**5**	**6**
High	68%	80%	67%	100%	64%	91%
Medium	29%	15%	28%	0%	33%	9%
Low	3%	5%	5%	0%	3%	0%

Comparison to related approaches

Precision and *recall* (Baeza-Yates & Ribeiro-Neto, 1999) are two commonly used metrics to evaluate the utility of traceability techniques (e.g., Spanoudakis et al., 2004). *Recall* is the percentage of all true links retrieved, and *precision* equals the percentage of true links in the answer set. Typically, both recall and precision are equally important. However, when it comes to dependency links, the recall is more important, as all dependency pairs must be found, in order to

[20] The users are able to specify the confidence level when associating a product fragment with a concept, i.e. high, medium or low, recall chapter 6.

avoid errors in the system under implementation. While precision is used as a filter, i.e. shows how many false positives need to be examined. Consequently, low precision rate will make automated linking useless.

Recall Table 8.2, the precision level for case 1 is 75% (see column 10), when taking into consideration partially correct. While precision of totally correct (column 9) is 57%[21]. The corresponding values for case 2 are 69% and 48%. The experiment does not lend itself to calculate recall, however. Firstly, the complete set of correct (true positive) dependency links was not known and we had no expert available. Secondly, as our method is user-centric, it was more important to observe users' perception of effectiveness. However, the earlier discussed amount of additional dependency links (correct and partial correct, column 8 in Table 8.2), identifies that recall using the prototype is higher than processing the fragments "manually". Actually, our method provides 100% recall since associating all product fragments together with domain concepts results in a network of indirectly linked product fragments.

There are many approaches tackling the problem of dependency links discovery with adoption of Information Retrieval (IR) techniques (e.g., Cerbah & Euzenat (2001); Spanoudakis et al. (2004)). Unfortunately, they support only natural language based fragments.

In addition, the special settings (i.e. distributed development) for which our method is developed and span of the whole development life-cycle by the method, makes it difficult to compare with other methods. Since other approaches mainly deal with a limited set of development life-cycle phase (i.e. limited set of product fragments types), e.g., requirements to architecture (Pohl et al., 2001). Some approaches are based on a particular tool (e.g., IBM® Rational Rose™ (Letelier, 2002) or specific notation family (e.g., UML (Knethen, 2002)).

8.5.3 Threats to validity

The following possible threats to the validity of this experiment have been identified.

- The case study is executed at the university. However, the experiment examined the real cases, 80% of the test subjects had an industrial experience.
- Different than intended usage settings. In intended settings, the user first participates in problem definition and later, he or she associates *own* product fragments (he or she has developed) with the concepts from domain model. Here the users were given the product fragments developed by others. Furthermore, domain models for both cases were not made by them.
- Fair answers vs. colleagues answers. The above analysis of the perceived usefulness construct shows that answers are consistent and very likely fair.

[21] Precision ratio of 50% means that developer has to examine about one false positive per true link.

- Subjects provided subjective evaluations. The individuals interpret the experimental materials and tasks according to their experience. Experience seemed to be similar for most of individuals.
- Subjective choice of comparative tools. Telelogic Doors is one of the leading tools in the area. Traceability matrix was chosen to use as one of the traditional techniques. Availability of IR techniques-based tools (e.g., (Spanoudakis et al., 2004)) is limited, since most of them are academic prototypes. Even if we would have such a tool, its applicability would be limited because of variety of product fragments types (i.e. binary files, see Appendix D).
- Fatigue effect. On average 3,5 hours were spent to complete the tasks and fill the questionnaire. Therefore, this effect is not relevant.

8.6 Satisfying the requirements

Here we revisit the requirements specified in chapter 2 (see section 2.6) and discuss how well they are satisfied by the proposed method and implemented prototype. For comparison with the state-of-the-art, recall Table 3.2.

Table 8.13 Meeting requirements by the method and prototype

Requirements	**Prototype/ Method**	**Comments**
Req1-Flexible access to the product	High	We do not use locking, developers are allowed to sign-out objects for revision.
Req2-Unrestricted product fragment types	High	Implementation of the method allows easy incorporation of new object types.
Req3-Unrestricted relation types	High	Possible to define own relations, kind of personal dependency relations.
Req4-Incremental product fragment refinement	High	We allow to post into repository fragments at any stage of development, temporary versions can also be loaded to repository.
Req5-Support for boundary objects	High	Different views on project and product fragments are possible, based on metadata, domain model, used categories, etc.
Req6-Active delivery of information	High	It is possible to subscribe for the notifications about different event.
Req7-Knowledge externalisation in a means of conceptual domain model	Medium	Only theoretical method is proposed, as modelling environment is not yet fully implemented.
Req8-Domain concepts explanation (extension)	High	Associations with product fragments are one example, as well description in

Requirements	**Prototype/ Method**	**Comments**
		natural language.
Req9-Support for knowledge internalization	Medium	Again, because of limitations of the prototype implementation, in particular absence of integrated modelling environment.
Req10-Conceptual domain model should be available through whole development life-cycle	High	It is available for association and for analysis. Conceptual model is used throughout development life-cycle and all product fragments can be associated.
Req11-Flexible metadata specification about development objects	High	Association with domain model was perceived ease to use, that one kind of metadata.
Req12-Efficient dependency management	High	Fuzzy association, and stepwise refinement to more explicit dependency links.

8.7 Summary

The feasibility of the proposed method and its implementation has been evaluated in this chapter. In the analysis we have focused on effectiveness and acceptance of the method (recall the evaluation questions of the experiment described in section 8.4.1). The results of the experiment clearly indicate the method being effective and helpful in dependency links discovery. The subjects' perceptions seem to confirm the performance-based results. The subjects perceived our method as ease to use and useful, and they expressed intention to use.

We have corroborated that the test subjects produced consistent answers. Though the scope was limited, results of the experiment give some credible indications on the method applicability and feasibility. Yet, the results should be interpreted only as preliminary, due to limited scope and amount of data, as well as artificial, different than intended, use of the method and tool. In intended settings, the user first participates in problem definition and later, he or she associates *own* product fragments (he or she has developed) with the concepts from domain model. Therefore, a larger scale experiment that would imitate the intended use of the method is necessary in order to reconfirm the results obtained. In addition, an experiment on known set of correct dependency links needs to be conducted.

Furthermore, the experiment has shown the necessity to improve the user interface of the current prototype. Telelogic Doors outperformed the studied prototype in visualisation of dependency links. Therefore, we need to consider enhancing manipulation of results by a means of providing different views and filters.

9 Conclusions and Outlook

"While we are free to choose our actions, we are not free to choose the consequences of our actions."
— Stephen R. Covey

This chapter concludes the thesis. The main contributions are outlined; advantages and limitation of the proposed method are discussed. Open challenges and possible further improvement are discussed at the end.

9.1 Summary of Contributions

Recall the overall objective defined for this thesis in chapter 1, i.e. *introduce a method for distributed collaborative work environment supporting management and change impact prediction of the diverse product fragments based on the semantics of the product fragments.*

In order to achieve the objective, we have proposed a method for change impact assessment and management in distributed development, based on conceptual domain model. The need for such method stems from the fact that in such projects system specification is produced in different formats and various tools might be used. Manual establishment of dependency links between product fragments is cumbersome in a distributed project. It is even difficult to keep an overview who is working on what in such settings. For instance, Egyed and Grunbacher (2005) report that requirements traceability in practice often suffers from the enormous effort and complexity of creating and maintaining traces. Meanwhile, fully automatic traceability linking is hardy achievable. That stems from NLP techniques being yet not so advanced and will require tailoring towards particular object types, as well different working style, i.e. socio-cultural distance between involved stakeholders. Consequently, we have proposed more fuzzy user-centric method to deal with complexity.

As prerequisites for the method a framework for model fragment management has been elucidated, repository support and implemented prototype have been discussed. In addition to being important part of our method, implementation of repository was supposed to be a kernel for the research activity in the IS-group at NTNU. In particular, the implementation targeted to provide a means for storing and manipulating the model fragments as supporting technology for the approaches developed earlier in the IS-group. First, an approach elucidated by Su (2004) – providing a storage for models (ontologies), extensions of concepts that are prerequisite for model mapping, and results of the produced mappings. Second, an approach to semantic document modelling and retrieval presented by Brasethvik (2004) – storing and reusing model fragments for document (information) retrieval.

Finally, the method and implementation have been evaluated in an experimental case study. The experience gained from the experiment, also supported by the results of the experiment, indicates that the proposed method is a promising line of research in the area of conceptual model (ontology) centric development. Meanwhile, Figure 9.1 summarises the proposed method by relating the goals, means to achieve them and abstract functional steps of the proposed method.

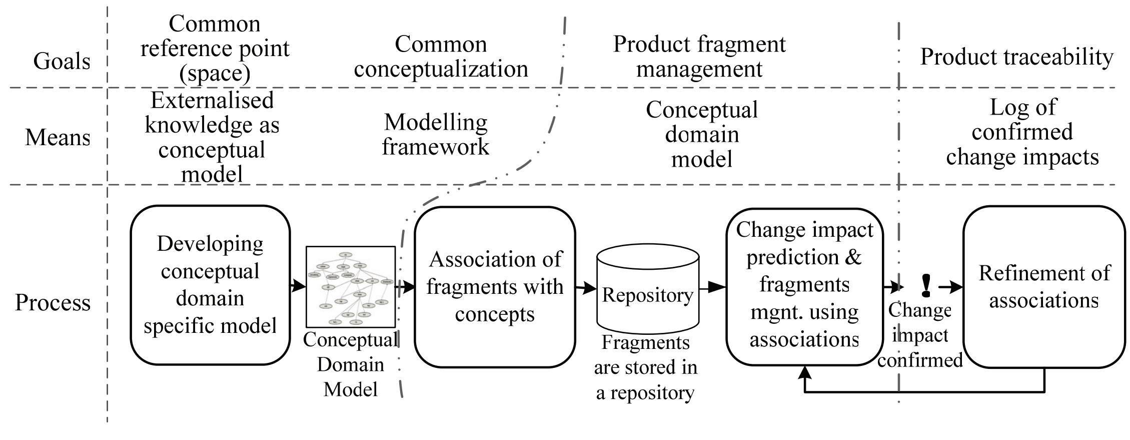

Goals
Common reference point (space)
Common conceptualization
Product fragment management
Product traceability
Means
Externalised knowledge as conceptual model
Modelling framework
Conceptual domain model
Log of confirmed change impacts
Process
Developing conceptual domain specific model
Conceptual Domain Model
Association of fragments with concepts
Repository
Fragments are stored in a repository
Change impact prediction & fragments mgnt. using associations
!
Change impact confirmed
Refinement of associations

Figure 9.1 Overview of the proposed method: goals, means and process

To sum up, the main advantages of the proposal are as follows.

- *Helps to reconcile terminology.* Stakeholders can agree to use a common vocabulary for the domain, or use own terminology, which is aligned to the common vocabulary. The chances of mutual understanding are greatly enhanced.
- *Facilitates an interpretation of development object.* Conceptual model used for (semantically) structuring development objects allows abstracting from the medium on which data is represented.

- *Facilitates change impact management in distributed heterogeneous environment*. As reported in chapter 8, test subjects agreed on the method being easier to use and helpful in discovery of dependency between product fragments.
- *Facilitates control of a project*. Amount of product fragments (recently) associated with particular concepts identifies where most of work is undergoing, i.e. what part of problem has a focus (currently). This allows project manager to distribute resources more equally or prioritise other parts.
- *Facilitates maintenance*. Especially when a new requirement appears, it is easier to associate with a domain concept and get an overview of possible impacts, then investigate a huge set of product fragments for possible dependencies.

9.2 Open Challenges

In this section we discuss the limitations of our method and give an account to possible extensions. Of course there are two high priority things to do. First, implement more reliable prototype, e.g., fix bugs, fully integrate modelling environment. Second, conduct real case study by either adopting the tool and method in an industrial project, or running several students projects. These two are important, but there are more open possibilities to improve the methodological and technological sides of the proposal. They are discussed in the following subsections.

9.2.1 Collaborative domain modelling

Here we propose to use conceptual models not only to guide the design of a system, but also to actually access and manage the information produced during IS engineering. Semantic associations of development objects with a concept from a domain model are intended to communicate the meaning of development objects between stakeholders. Since success in system development depends on effective human communication (Solvberg & Kung, 1993), early understanding and modelling of problem domain is a key to managing large scale systems and projects. This requires stakeholders to reach certain level of shared interpretation of the domain referred throughout the development. Within a limited domain, it is possible to engage stakeholders in a collaborative activity to explicitly define the semantics of the domain.

However, the limited domains usually are not targeted by the huge distributed projects. That is one of the most significant issues, i.e. the need to establish a common underlying structure that provides communication, interaction and management between the different parties engaged in the systems development, as well as overcoming differences in terminology, opinion, expertise and understanding of the domain. Hoppenbrouwers et al. (2005b) states that "*most actual modelling is done by individuals, two people at most. Genuine group modelling sessions are very rare*". Though, they do not provide any details on what scale projects they have investigated. Therefore, there is an obvious need to conduct more empirical studies on how modelling is actually performed. It is dangerous for method applicability, if only experts are allowed to contribute to the models.

Fortunately, there are some evidences of collaborative problem conceptualisation being possible. Zhdanova et al. (2005) proposes community-driven ontology management, i.e., providing means to the community members to develop and maintain domain ontologies and to crosslink between different domains. They hypothesis is similar to our, i.e., "the ontologies which are constructed, aligned and further operated by the communities *represent* the *domain* and connection with other domains *more comprehensibly* than the ontologies designed and maintained by an external knowledge engineer". Zhdanova et al. (2005) prove that in a case study of creating a portal ontology, i.e. results indicate that experts are not capable to specify the community knowledge comprehensively, as a community would do it itself.

Furthermore, a common model is important for developers' motivation, they do not feel "boxed", but have an overall view how do they fit into the whole, e.g., Hoppenbrouwers et al. (2005b) observe that "models are particularly important in giving stakeholders a feeling that they are "part of the larger whole". Often, just knowing where in the model "they can be found" is important to stakeholders, even if they do not understand the fine points of the model." Therefore, it would be interesting to investigate whether the method is applicable in open source software development communities, as these projects are widely distributed. Though, it might be difficult to get developers into the modelling phase, as they usually are programmers, not modellers. In addition, developers typically work on multiple projects at different levels of involvement in open source projects (Barnett, 2004).

9.2.2 Towards automation

Natural language processing (NLP) is an active and steady improving research area. Researchers have used NLP techniques to generate structured or formal models from requirements documents expressed in natural language, or to identify terms denoting significant entities that deserve further consideration and analysis within such documents.

Adoption of NLP techniques should be considered for both, domain model construction and product fragment analysis for the purpose of associating with the domain model. In these both areas there are already some contributions. For instance, model generation from requirement documents has been based on semantic or grammatical analysis of natural language. Examples of systems advocating the semantic analysis approach include CICO (Ambriolla & Gervazi, 1997), OICSI (Rolland & Proix, 1992), COLOR-X (Burg & van de Riet, 1995), Moreno (1998), LIDA system (Overmyer et al., 2001), and Text2onto (Cimiano & Volker, 2005).

Actually, earlier mentioned CnS system and approach by Brasethvik (2004) consider both, domain model construction based on document collection and later, linguistic analysis of new documents in order to propose relevant model fragments in terms of selected domain model concepts and named relations.

However, automation of domain model construction most likely will make it even harder to comprehend the model, commit to and use it. Therefore, adoption of NLP techniques for semi-automatic association of product fragments to domain concepts is more promising. Unfortunately, it is still troublesome to automate analysis code fragments, where results mostly depend on ones programming habits (coding and naming classes, functions, etc). Though there are some attempts, e.g., Antoniol et al. (2002) have proposed the use of information retrieval techniques to support the generation of traceability relations between requirement documents and source code. Stepwise refinement of conceptual model and inclusion of design/ implementation specific concepts would facilitate association of technical product fragments, i.e. code fragments, design diagrams.

Having a robust automatic association of product fragments, would provide a good means for controlling whether the system (solution) is within boundaries of problem space (UoD). For instance, having bigger domain model and deciding to computerise only a part of it, then all product fragments associated with concept outside of to-be-computerised area of the problem domain, can be treated as declination from the target. Therefore, a solution is going to be not feasible.

As we have already mentioned importance of the fourth dimension in collaborative modelling (see chapter 5). The representation dimension would traverse a conceptualisation of problem starting informal, natural language descriptions and reaching more formal definition. The strength of including this dimension lies in providing the means for stakeholders without a modelling experience to be part of the process. Seamless transition from informal natural language descriptions of UoD to formal (semi-formal) models would make modelling easier for them. Furthermore, adoption of model explanation generation (Gulla, 1993) techniques may help to comprehend the model fragments. Relating documents (kind of governing documents, describing the domain) to concepts (as it is proposed by Brasethvik (2004)) helps to explain the model as well as validate the model, facilitating agreement and commitment. Inclusion of representation dimension will be beneficial, though it preferably requires adoption of NLP techniques, discussed earlier in this section. However, supporting modelling by various notations is a separate interesting research area along the representation axis and needs to be investigated further.

9.2.3 Other improvements

Conceptual domain model in our method is used to relate all product fragments. Given such relational network of interconnected product fragments; it is natural to think about reuse of product fragments based on similarity (equality) of problem domain. For instance, Feature-Oriented Domain Analysis (FODA) focuses on the systematic discovery and exploitation of commonality and variability in related software systems (Kang et al., 1990) or reuse by domain analogy as proposed by Sutcliffe and Maiden (1998). FODA is primarily used to identify distinct features in the domain. Then it is possible to extract suitable product fragments based on matched (overlapping) part of conceptual domain model. In order to make it efficient, domain model should be represented in a reasoning enabled language, e.g., OWL (Web Ontology Language) (W3C, 2005). Then reasoning process will rely more on model properties, not only computed weights between concepts.

Finally, for successful adoption in practice, the system's functionality needs to be radically increased, for instance, workflow engine should be included, and system should be portable for a Web-browser.

Bibliography

(Agerfalk et al., 2005) Ågerfalk, P.J., Fitzgerald, B., Holmström, H., Lings, B., Lundell, B., and Ó Conchúir, E. A Framework for Considering Opportunities and Threats in Distributed Software Development, In *Proc. of the Intl. Workshop on Distributed Software Development* (DiSD 2005), Paris, 2005.

(Andersen, 1994) Andersen, R. *A Configuration Management Approach for Supporting Cooperative Information System Development*, PhD thesis, NTH, Trondheim, Norway, 1994.

(Amber, 2002) Amber, S.W. *Agile Modeling: Effective Practices for Extreme Programming and the Unified Process*, John Wiley & Sons., 2002, 384 p.

(Ambriolla & Gervazi, 1997) Ambriolla, V., and Gervazi, V. Processing natural language requirements. In *Proc. of Intl. Conf. in Automated Software Engineering* (ASE '97), pages 36–45.

(Antoniol et al., 2002) Antoniol, G., Canfora, G., Casazza, G., De Lucia, A., and Merlo, E. Recovering traceability links between code and documentation. *IEEE Transactions on Software Engineering* 28(10), 2002, pages 970–983.

(Avison & Fitzgerald, 2002) Avison, D.E., and Fitzgerald, G. *Information Systems Development. Methodologies, Techniques and Tools*. 3rd edition. McGraw-Hill, 2002, 608 p.

(Baeza-Yates & Ribeiro-Neto, 1999) Baeza-Yates, R., and Ribeiro-Neto, B. *Modern Information Retrieval*. Addison Wesley, ACM Press, 1999, 513 p.

(Bannon & Bodker, 1997) Bannon, L., and Bødker, S. Constructing Common Information Spaces. In *Proc. of 5th European Conf. on CSCW*. Lancaster, UK. Kluwer Academic Publishers, 1997.

(Barnett, 2004) Barnett, L. Applying Open Source Processes in Corporate Development Organizations. Forrester Research, Inc. 2004.

(Berners-Lee et al., 2001) Berners-Lee, T., Handler, J., and Lassila, O. The Semantic Web. *Scientific American*, May 2001, pages 34–43.

(Bernstein & Dayal, 1994) Bernstein, P.A., and Dayal, U. An Overview of Repository Technology. In *Proc. of 20th VLDB conference*, 1994, pages 705–713.

(Bleeker et al., 2004) Bleeker, A.I., Proper, H.A., and Hoppenbrouwers, S.J.B.A. The Role of Concept Management in System Development - A practical and a theoretical perspective. In Grabis, J., Persson, A., and Stirna, J. (Eds.), *Forum proc. of the 16th Conf. on Advanced Information Systems 2004* (CAiSE 2004), Riga Technical University, Riga, Latvia, 2004, ISBN 998497670X, pages 73-82.

(Boehm, 1987) Boehm, B.W. A spiral model of software development and enhancement. In R.H. Thayer (Ed.) *Tutorial: Software Engineering Project Management*, IEEE Computer Society, Washington, 1987, pages 128–142.

(Borgman, 2000) Borgman, C.L. *From Gutenberg to the Global Information Infrastructure: Access to Information in the Networked World*. MIT Press, 2000, 324 p.

(Brasethvik, 2004) Brasethvik, T. *Conceptual modelling for domain specific document description and retrieval- An approach to semantic document modelling*. PhD thesis, IDI, Norwegian University of Science and Technology (NTNU), Trondheim, Norway, 2004.

(Brown et al., 1992) Brown, A.W., Earl, A.N., and McDermid, J.A. *Software Engineering Environments – Automated Support for Software Engineering*, McGraw-Hill, London, 1992, 326 p.

(Bu et al., 2001) Bu J., Lin Q., Ng J., and Low C.P. VRCASE: A Virtual Reality Based Collaborative CASE Tool. In *Proc. of IASTED Intl. Conf., Applied Informatics* (AI 2001), February 19-22, 2001, Innsbruck, Austria.

(Budgen et al., 1993) Budgen, D., Marashi, M., and Reeves, M. CASE tools: Masters or servants? In *Proc. of the 1993 Software Engineering Environments Conference*. IEEE Computer Society Press, pages 156–165.

(Bunge, 1998) Bunge, M.A. *The philosophy of science*. Transaction publishers, USA, 1998.

(Bunge, 1977) Bunge, M.A. *Ontology I: The Furniture of the World*, *vol.3 of Treatise on Basic Philosophy*. Reidel, Boston, 1997.

(Burg & van de Riet, 1995) Burg, J.F.M., and van de Riet, R.P. COLOR-X: Linguistically-based Event Modeling: A General Approach to Dynamic Modeling. In Iivari, J, Lyytinen, K., Rossi, M. (Eds.) *Proc. of 7th Intl. Conf. on Advanced Information Systems Engineering*, (CAiSE'95), LNCS 932, Springer-Verlag, 1995, pages 26-39.

(Carlsen, 1997) Carlsen, S. *Conceptual Modelling and Composition of flexible workflow models*. PhD thesis, Norwegian University of Science and Technology (NTNU), Trondheim, Norway, 1997.

(Carstensen & Schmidt, 2002) Carstensen, P.H., and Schmidt, K. Self-Governing Production Groups: Towards Requirements for IT Support. In *Proc. of the IFIP TC5/WG5.3 5th IFIP/IEEE Intl. Conf. on Information Technology for Balanced Automation Systems in Manufacturing and Services: Knowledge and Technology Integration in Production and Services: Balancing Knowledge in Product and Service Life Cycle*, Kluwer, 2002, pages 49-60 .

(CEEBI, 2004) Centre for Extended Enterprise and Business Intelligence. Multisite Software Engineering – Research Programme, Curtin University of Technology, Australia. URL: http://www.ceebi.research.cbs.curtin.edu.au/docs/RES_multiSE.php (Last checked: 2005 09 13).

(Cerbah & Euzenat, 2001) Cerbah, F., and Euzenat, J. Traceability between models and texts through terminology. *Data and Knowledge Engineering* 38 (1), Elsevier Science Publishers, 2001, pages 31-43.

(Chalupsky, 2000) Chalupsky, H. Ontomorph: A translation system for symbolic logic. In Cohn, A.G., Giunchiglia, F., and Selman, B. (Eds.), *KR2000: Principles of Knowledge Representation and Reasoning*, San Francisco, California, USA, 2000. Morgan Kaufmann, pages 471–482

(Chandrasekaran et al., 1999) Chandrasekaran, B., Josephson, J.R., and Benjamins, V.R. What are ontologies, and why do we need them? *IEEE Intelligent Systems* 14(1), Jan./Feb. 1999, pages 20–26.

(Chen & Norman, 1992) Chen, M., and Norman, R. A Framework for Integrated CASE. *IEEE Software* 3*,* 1992, pages 18–22.

(Chen et al., 1999) Chen, P.P., Thalheim, B., and Wong, L.Y. Future Directions of Conceptual Modeling. In Chen, P.P., Akoka, J., Kangassalo, H., Thalheim, B. (Eds.) *Conceptual Modeling, Current Issues and Future Directions, Selected Papers from the Symposium on Conceptual Modeling*, Los Angeles, California, USA, held before ER'97. LNCS 1565, Springer-Verlag, 1999, pages 287-301.

(Cimiano & Volker, 2005) Cimiano, Ph., and Völker, J. Text2Onto - A Framework for Ontology Learning and Data-Driven Change Discovery. In Montoyo, A., Muñoz, R., and Métais, E. (Eds.) *Proc. of 10th Intl. Conf. on Applications of Natural Language to Information Systems (NLDB'05).* LNCS 3513, Springer-Verlag, 2005, pages 227-238.

(Cleland-Huang et al., 2003) Cleland-Huang, J., Chang, C.K., and Wise, J. Automating Performance Related Impact Analysis through Event Based Traceability. *Requirements Engineering Journal* 8(3), Springer-Verlag, Aug. 2003, pages 171-182.

(Composent, 2005) Composent Collaboration Plugin. URL: http://www.eclipseplugincentral.com/ displayarticle172.html. (Last checked: 2005 12 28).

(Conradi & Westfechtel, 1998) Conradi, R., and Westfechtel, B. Version Models for Software Configuration Management. *ACM Computing Survey*s 30(2), 1998, pages 232-282.

(Cormen et al., 2001) Cormen, T.H., Leiserson, C.E., Rivest, R.L., and Stein, C. *Introduction to Algorithms*, 2nd Edition. The MIT Press and McGraw-Hill, 2001.

(Cramer, 1994) Cramer, D. *Introducing Statistics for Social Research: Step-by-step calculations and computer techniques using SPSS*. London: Routledge, 1994.

(CWM, 2005) Data Warehousing, CWM, and MOF Resource Page. URL: http://www.omg.org/technology/cwm/ (Last checked: 2005 12 12)

(Daconta et al., 2003) Daconta, M.C., Orbst, L.J., and Smith, K.T. *The Semantic Web*, John Wiley & Sons, 2003, 312 p.

(Daft, 1995) Daft, R.L. *Organization theory & design*. St. Paul: West publishing Company, 1995, 511 p. ISBN 0-314-04452.

(Dart, 1991) Dart, S. Concepts in configuration management systems. In Feiler, P.H. (Ed.), *Proc. of 3rd Intl. Workshop on Software Configuration Management*, Trondheim, Norway, June 1991, ACM Press, pages 1–18.

(Davis, 1989) Davis, F.D. Perceived usefulness, perceived ease of use and user acceptance of information technology. *MIS Quaterly* 13(3), 1989, pages 319-340.

(Decker et al., 1999) Decker, S., Erdmann, M., Fensel, D. and Studer, R. Ontobroker: Ontology based Access to Distributed and Semi-Structured Information. In *Semantic Issues in Multimedia Systems*. Kluwer Academic Publisher. 1999.

(Devedzic, 2002) Devedžić, V. Understanding Ontological Engineering. *Communications of the ACM* 45(4), pages 136-144, April 2002.

(Dijkstra, 1959) Dijkstra, E.W. A note on two problems in connexion with graphs. *Numerische Mathematik* 1, 1959, pages 269–271.

(Dillenbourg et al., 1996) Dillenbourg, P., Baker, M., Blaye, A., and O'malley, C. The evolution of research on collaborative learning. In Spada, E., and Reiman, P. (Eds.) *Learning in Humans and Machine: Towards an interdisciplinary learning science*. Oxford: Elsevier, 1996, pages 189-211.

(ebXML, 2005) ebXML – Enabling a global electronic market. URL: http://www.ebxml.org/ (Last checked: 2005 12 12)

(Eclipse, 2005) Eclipse platform. http://www.eclipse.org/. (Last checked: 2005 08 22).

(Egyed, 2003) Egyed, A. Compositional and relational reasoning during class abstraction. In *Proc. of the 6th Intl. Conf. on the Unified Modeling Language* (UML), San Francisco, USA, 2003, pages 121-137.

(Egyed, 2001) Egyed, A. A Scenario-Driven Approach to Traceability. In *Proc. of the 23rd Intl. Conf. on Software Engineering* (ICSE), Toronto, Canada, 2001.

(Egyed & Grunbacher, 2005) Egyed, A., and Grünbacher, P. Supporting Software Understanding with Automated Requirements Traceability. *Journal of Software Engineering and Knowledge Engineering* (JSEKE), 2005, in press.

(Egyed & Kruchten, 1999) Egyed, A., and Kruchten, P. Rose/Architect: a tool to visualize architecture. In *Proc. of the 32nd Hawaii Intl. Conf. on System Sciences* (HICSS), 1999.

(Ellis et al., 1991) Ellis, C.A., Gibbs, S.J., and Rein, G.L. Groupware some issues and experiences. *Communications of the ACM* 34(1), 1991, pages 38-59.

(Erichsen, 2003) Erichsen, K.O. *Enabled Traceability in Distributed System Development*. Master thesis, IDI, Norwegian University of Science and Technology (NTNU), Trondheim, Norway, 2003.

(Estublier, 2001) Estublier, J. Objects control for software configuration management. In Dittrich, K.R., Geppert, A., and Norrie, M.C. (Eds.) *Advanced Information Systems Engineering, proceedings of 13th Intl. Conf. on Advanced Information Systems Engineering* (CAiSE'2001), Interlaken, Switzerland, LNCS 2068, Springer-Verlag, 2001, pages 359-373.

(Estublier & Casallas, 1995) Estublier, J., and Casallas, R. Three dimensional versioning. In Estublier, J. (Ed.), *Software Configuration Management: selected papers* / ICSE SCM-4 and SCM-5 workshops, Springer-Verlag, LNCS 1005, Seattle, Washington, October 1995, pages 118–135.

(Falkenberg et al., 1997) Falkenberg, E.D., Hesse, W., Lindgreen, P., Nilsson, B.E., Han Oei, J.L., Rolland, C., Stamper, R.K., van Asche, F.J.M., Verrjin-Stuart, A., Voss, K. FRISCO - A Framework of Information Systems Concepts. *IFIP WG 8.1 Technical Report* – IFIP WG 8.1 Task Group FRISCO, 1997.

(Farshchian, 2001) Farshchian, B.A. *A Framework for Supporting Shared Interaction in Distributed Product Development Projects*. PhD thesis, IDI, Norwegian University of Science and Technology (NTNU), Trondheim, Norway, 2001.

(Faulkner, 2003) Faulkner, L. Beyond the five-user assumption: Benefits of increased sample sizes in usability testing. *Behavior Research Methods, Instruments, & Computers* 35(3), 2003, pages 379-383.

(Fellbaum, 1998) Fellbaum, C. *WordNet: An Electronic Lexical Database*. MIT Press, 1998.

(Fensel, 2001) Fensel, D. *Ontologies: A Silver Bullet for Knowledge Management and Electronic Commerce*. Springer-Verlag, 2001, 147 p.

(Fidjestol, 2005) Fidjestøl, A.D. *An Editor with Repository Support for Conceptual Modeling in Information System Design*. Master thesis, Norwegian University of Science and Technology (NTNU), Trondheim, Norway, 2005.

(Finkelstein et al., 1991) Finkelstein, A., Kramer, J., Nuseibeh, B., Finkelstein, L., and Goedicke, M. Viewpoints: a framework for integrating multiple perspectives in system development. *Intl. Journal of Software Engineering and Knowledge Engineering* 2(1), 1991, pages 31–58.

(Frezza et al., 1996) Frezza, S.T., Levitan, S.P., and Chrysanthis, P.K. Linking requirements and design data for automated functional evaluation. *Computers in Industry*, 30(1), September 1996, pages 13–25.

(Fuggetta, 1993) Fuggetta, A. A classification of CASE technology. *Computer* 26(12), December 1993, pages 25-38.

(Gamma et al., 1995) Gamma, E., Helm, R., Johnson, R., and Vlissides, J. *Design Patterns - Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995, 395 p.

(Gjersvik, 1993) Gjersvik, R. *The Construction of Information Systems in Organization: An Action Research Project on Technology, Organizational Closure, Reflection, and Change.* PhD thesis, NTH, Trondheim, Norway, 1993.

(Glossary, 2005) URL: http://uis.georgetown.edu/departments/eets/dw/GLOSSARY0816.html#R (Last checked: 2005 10 03)

(Gray & Ryan, 1997) Gray, J.P., and Ryan, B. Applying the CDIF standard in the construction of CASE design tools. In Bailes, P. (Ed.), *Proc. of the Australian Software Engineering Conf.*, IEEE, 1997, pages 88–97.

(Gruber, 1991) Gruber, T.R. The Role of Common Ontology in Achieving Sharable, Reusable Knowledge Bases. In Allen, J., Fikes, R., and Sandewall, E. (Eds.), *Principles of Knowledge Representation and Reasoning*, Morgan Kaufman, San Mateo, CA, 1991.

(Gruber, 1993) Gruber, T.R. A Translation Approach to Portable Ontology Specifications. *Knowledge Acquisition* 5(2). June 1993, pages 199-220.

(Grunbacher et al., 2001) Grünbacher, P., Egyed, A., and Medvidovic, N. Reconciling software requirements and architectures - the CBSP approach. In *Proc. of the 5th IEEE Intl. Symposium on Requirements Engineering* (RE'01), 2001, pages 202–211.

(Gruninger & Lee, 2002) Gruninger, M., and Lee, J. Ontology applications and design: Introduction. *Communications of the ACM* 45(2), 2002: Special Issue: Ontology applications and design, pages 39-41.

(Guarino, 1998) Guarino, N. Formal Ontology and Information Systems. In Guarino, N., (Ed.) *Proc. of the 1st Intl. Conf. on Formal Ontologies in Information Systems* (FOIS'98), IOS Press, June 1998, pages 3-15.

(Guarino & Poli, 1995) Guarino, N., and Poli, R. Editorial: The role of formal ontology in the information technology. *Intl. Journal of Human-Computer Studies* 43(5-6), 1995, pages 623-624.

(Gulla, 1993) Gulla, J.A. *Explanation Generation in Information Systems Engineering*. PhD thesis, NTH, Trondheim, Norway, 1993.

(Gulla et al., 1991) Gulla, J.A., Lindland, O.I., and Willumsen, G. PPP - an integrated case environment. In *Proc. of the 3rd Intl. Conf. on Advanced Information Systems Engineering* (CAiSE'91), Trondheim, Norway, Springer-Verlag, LNCS 498, 1991, pages 194-221.

(Gustas & Gustiene, 2003) Gustas, R., and Gustiene, P. Towards the Enterprise engineering approach for Information system modelling across organisational and technical boundaries. In *Proc. of the 5th Intl. Conf. on Enterprise Information Systems, vol. 3*, Angers, France, 2003, pages 77-88.

(Halpin, 2001) Halpin, T. *Information Modeling and Relational Databases, From Conceptual Analysis to Logical Design*. Morgan Kaufman, San Mateo, California, USA, 2001, 792 p.

(Hatch, 1997) Hatch, M.J. *Organization theory. Modern, symbolic, and postmodern perspectives*. New York: Oxford university press, 1997. 379 p. ISBN 0-19-877491-5.

(Henriksen et al., 1997) Henriksen, T.R., Fidjestøl, A.D., and Aubert, A.B. PPP Repository Management, Technical report (ver. 15th of October 1997), IDI, NTNU, Trondheim, Norway, 1997.

(Hoppenbrouwers et al., 2005a) Hoppenbrouwers, S.J.B.A., Bleeker, A.I., and Proper, H.A. Facing the Conceptual Complexities in Business Domain Modeling. *Computing Letters* 2 (1), 2005, pages 59-68.

(Hoppenbrouwers et al., 2005b) Hoppenbrouwers, S.J.B.A., Proper, H.A., and van der Weide, Th.P. Understanding the Requirements on Modelling Techniques. In Pastor, O., and Falcão e Cunha, J. (Eds.)*, Proc. of 17th Intl. Conf. on Advanced Information Systems Engineering* (CAiSE'2005). LNCS 3520, Springer-Verlag, 2005, pages 262-276.

(Huisman & Iivari, 2002) Huisman, M., and Iivari, J. The individual deployment of systems development methodologies. In A. Banks Pidduck et al. (Eds.) *Proc. of 14th Intl. Conf. on Advanced Information Systems Engineering* (CAiSE'2002), LNCS 2348, Springer-Verlag, 2002, pages 134-150.

(IBM, 2005a) IBM. IBM Rational Software. URL: http://www-306.ibm.com/software/rational/. Last checked: (2005 10 05)

(IBM, 2005b) IBM. Rational Unified Process. URL: http://www-306.ibm.com/software/awdtools/rup/. Last checked: (2005 10 05)

(IBM, 1996) IBM. *Developing object-oriented software: an experience-based approach*. Prentice Hall, Inc., Upper Saddle River, NJ, 1996, 636 p. ISBN: 0137372485

(IEEE, 1990) IEEE. *IEEE Standard Glossary of Software Engineering Terminology*. IEEE, New York, September 1990.

(IEEE, 1988) The Institute of Electrical and Electronics Engineers, Inc., New York. IEEE Guide to Software Configuration Management, 1988. *ANSI/IEEE Standard 1042-1987*.

(Iivari, 1996) Iivari, J. Why are CASE Tools Not Used? *Communications of the ACM* 39, 1996, pages 94-103.

(Jackson, 1990) Jackson, M. Some complexities in computer-based systems and their implications for system development. In *Proc. of Intl. Conf. on Computer Systems and Software Engineering* (CompEuro'90), Tel-Aviv, Israel, May 1990, pages 344–351.

(Jarzabek & Huang, 1998) Jarzabek, S., and Huang, R. The Case for User-Centered CASE Tools. *Communications of the ACM* 41(8), 1998, pages 93-99.

(Jasper & Uschold, 1999) Jasper, R., and Uschold, M. A Framework for Understanding and Classifying Ontology Applications. In *Proc. of 12th Workshop on Knowledge Acquisition Modeling and Management* (KAW'99), 1999.

(Kang et al., 1990) Kang, K.C., Cohon, S.G., Hess, J.A., Novak, W.E., and Peterson, A.S. Feature-Oriented Domain Analysis (FODA): Feasibility Study. Software Engineering Institute, Carnegie Mellon University, Pittsburgh, CMU/SEI-90-TR-21, November 1990.

(Kashyap & Sheth, 1994) Kashyap, V. and Sheth, A. Semantics-based Information Brokering. In *Proc. of the 3rd Intl. Conf. on Information and Knowledge Management* (CIKM), 1994, pages 363-370.

(Kelly et al., 1996) Kelly, S., Lyytinen, K., Rossi, M. MetaEdit+ A Fully Configurable Multi-User and Multi-Tool CASE and CAME Environment. In *Proc. of 8th Intl. Conf. on Advanced Information Systems Engineering (CAiSE'96)*, Heraklion, Greece, May 1996, Springer-Verlag, LNCS 1080, pages 1-21.

(Kemerer, 1992) Kemerer, C. F. How the Learning Curve Affects CASE Tool Adoption. *IEEE Software*, 9, 1992, pages 23-28.

(Kiely & Fitzgerald, 2003) Kiely, G., and Fitzgerald, B. An investigation of the use of methods within information systems development projects. In *Proc. of IFIP WG 8.2 Conference*, Athens, Greece, June 2003, pages 187-198.

(Knethen, 2002) von Knethen, A. Change-Oriented Requirements Traceability: Support for Evolution of Embedded Systems. In *Proc. of 18th Intl. Conf. on Software Maintenance* (ICSM 2002), Montréal, Canada, IEEE Computer Society 2002, pages 482-485.

(Krogstie, 1995) Krogstie, J. *Conceptual Modelling for Computerized Information Systems Support in Organizations*. PhD thesis, IDT, NTH, Trondheim, Norway, 1995

(Krogstie & Solvberg, 2000) Krogstie, J., and Sølvberg, A. *Information Systems Engineering: Conceptual Modeling in a Quality Perspective*, 2000. Norwegian University of Science and Technology, Trondheim (NTNU), Norway. Unpublished book.

(Krogstie et al., 2006) Krogstie, J., Veres, C., and Sindre, G. Interoperability through integrating Semantic Web Technology, Web Services, and Workflow Modeling. In Konstantas, D., Bourrières, J.-P., Léonard, M., and Boudjlida, N. (Eds.) *Interoperability of Enterprise Software and Applications*. Springer, 2006, pages 147-159.

(Kung & Solvberg, 1986) Kung, D.C., and Sølvberg, A. Activity modelling and behaviour modelling. In Olle, T., Sol, H., and Verrijn-Stuart, A. (Eds.), *Information System Design Methodologies: Improving the Practice*. Amsterdam, 1986.

(Lau, 2004) Lau, S. *Semantics based project content management*. Master thesis, IDI, Norwegian University of Science and Technology (NTNU), Trondheim, Norway, 2004.

(Lee et al., 2001) Lee, B.G., Narayanan, N.H., and Chang, K.H. An integrated approach to distributed version management and role-based access control in computer supported collaborative writing. *The Journal of System and Software* 59(2), 2001, Elsevier Science Publishers, pages 119-134.

(Letelier, 2002) Letelier, P. A framework for requirements traceability in UML based projects. In *Proc. of the 1st Intl. Workshop on Traceability*, Edinburgh, Scotland, UK, September 2002, pages 32–41.

(Libresource, 2005) LibreSource community. URL: http://dev.libresource.org/home. (Last checked: 2005 09 29)

(Likert, 1931) Likert, R. A technique for the measurement of attitudes. *Archives of Psychology*. New York, Columbia University Press, 1931.

(Loucoploulos & Karakostas, 1995) Loucopoulos, P., and Karakostas, V. *System Requirements Engineering*. McGraw Hill, 1995, 160 p.

(Loucopoulos & Champion, 1988) Loucopoulos, P., and Champion, R. Knowledge-based approach to requirements engineering using method and domain knowledge. *Knowledge-Based Systems* 1(3), June 1988, pages 179-187.

(Lundell & Lings, 2004) Lundell, B., and Lings, B. Changing perceptions of CASE technology. *Journal of Systems and Software* 72, 2004, Elsevier, pages 271-280.

(Mahler, 1994) Mahler, A. Variants: Keeping things together and telling them apart. In chapter 3 of Tichy, W.F. (Ed.) *Configuration Management, volume 2 of Trends in Software*. John Wiley & Sons, Chichester, UK, 1994, pages 39–69.

(Mathes, 2004) Mathes, A. Folksonomies – Cooperative Classification and Communication Through Shared Metadata. URL: http://www.adammathes.com/academic/computer-mediated-communication/folksonomies.pdf. (Last cheked: 2005 10 02)

(Matulevicius et al., 2004) Matulevičius R., et al. MEIS system requirements specification. Technical report, Norwegian University of Science and Technology (NTNU), 2004.

(Mayr, 2002) Mayr, H.C. Do We Need an Ontology of Ontologies? In Spaccapietra, C., March, S.T., Kambayashi, Y. (Eds.), *Proc. of 21st Intl. Conf. on Conceptual Modeling*, Tampere, Finland, October 7-11, 2002, LNCS 2503, Springer-Verlag, 2002, page 15.

(MDA, 2003) MDA guide version 1.0.1. Technical Report omg/2003-06-01, OMG, June 2003.

(Mena et al., 1996) Mena, E., Kashyap, V., Illarramendi, A., and Sheth, A. Managing multiple information sources through ontologies: Relationship between vocabulary heterogeneity and loss of information. In *Proc. of Knowledge Representation Meets Databases* (KRDB'96), ECAI'96 conference, August 1996, pages 50-52.

(Metis, 2005) Metis by Troux. URL: http://www.troux.com/products/metis/ (Last checked: 2006 01 15)

(Microsoft, 2005) Microsoft. Meta Data Services Overview. URL: http://msdn.microsoft.com/library/default.asp?url=/library/en-us/reposad/raoverview_3t87.asp. (Last checked: 2005 08 22)

(Microsoft/XIF, 1999) Microsoft Corp. Repository SDK 2.1b Documentation: XML Interchange Format (XIF). Microsoft Corp., Redmond, USA, May 1999.

(Miller et al., 2005) Miller, E., Swick, R., and Brickley, D. Resource Description Framework (RDF). URL: http://www.w3.org/RDF/ (Last checked: 2005 12 12)

(Miller, 2002) Miller, G.A. Wizard of the new wordsmiths: His idea to link words rewrote the dictionary. *The star ledger*, January 2002.

(Moody, 2001) Moody, D.L. *Dealing with Complexity: A Practical Method for Representing Large Entity Relationship Models*. PhD thesis, Department of Information Systems, University of Melbourne, Melbourne, Australia, 2001, 354p.

(Moreno, 1998) Moreno, A.M. Results of the Application of a Linguistic Approach to Object-Oriented Analysis. *Journal of Software Engineering and Knowledge Engineering* 8(4), 1998. pages 449-459.

(Moriarty, 2000) Moriarty, T. The importance of names, *The Data Administration Newsletter* 15, (2000).

(MS Access, 2003) Microsoft Office Online: Access 2003 Home Page. URL: http://office.microsoft.com/en-us/FX010857911033.aspx (Last checked: 2005 09 29)

(Mylopoulos, 1992) Mylopoulos, J. Conceptual modeling and Telos. Chapter 2 In Loucopoulos, P. and Zicari, R. (Eds.) *Conceptual Modeling, Databases, and CASE*, John Wiley & Sons, 1992, pages 49-68.

(MySQL, 2005) MySQL: The world's most popular open source database. http://dev.mysql.com/. (Last checked: 2005 09 29)

(Nielsen, 1993) Nielsen, J. *Usability Engineering*. Academic Press, Boston, 1993.

(Noy & Musen, 2000) Noy, N.F., and Musen, M.A. PROMPT: Algorithm and Tool for Ontology Merging and Alignment. In *Proc. of the National Conf. on Artificial Intelligence* (AAAI), 2000, pages 450-455.

(Nuseibeh, 2001) Nuseibeh, B. Weaving Together Requirements and Architectures. *IEEE Computer* 34(2), 2001, pages 115-117.

(Ogden & Richards, 1923) Ogden, C.K., and Richards, I.A. *The Meaning of Meaning*. 8th Edition, New York, Harcourt, Brace & World, Inc., 1923.

(Olive, 2005) Olive, A. Conceptual Schema-Centric Development: A Grand Challenge for Information Systems Research. In Pastor, O., and Falcão e Cunha, J. (Eds.) *Proc. of 17th Intl. Conf. Advanced Information Systems Engineering* (CAiSE'2005). LNCS 3520, Springer-Verlag, 2005, pages 1-15.

(Oliveira et al., 2004) de Oliveira, K.M., Zlot, F., Rocha, A.R., Travassos, G.H., Galotta, C., de Menezes, C.S. Domain-oriented software development environment. *Journal of Systems and Software* 72(2), 2004, pages 145-161.

(Olle et al., 1988) Olle, T.W., Hagelstein, J., MacDonald, I.G., Rolland, C., Sol, H.G., van Assche, F.J.M., Verrijn-Stuart, A.A. *Information Systems Methodologies: A Framework for Understanding*, Addison-Wesley, 1988.

(OMG/XMI, 1998) OMG. XML Metadata Interchange (XMI), Document ad/98-10-05, October 1998.

(Ophelia, 2003) The Ophelia project. URL: http://www.opheliadev.org. (Last checked: 2005 08 22)

(OpenShore, 2005) OpenSHORE. http://www.openshore.org/. (Last checked: 2005 12 28).

(Oracle, 2005) Oracle database. URL: http://www.oracle.com/database/. (Last checked: 2005 09 29)

(Overmyer et al., 2001) Overmyer, S., Lavoie, B., and Rambow, O. Conceptual Modeling through Linguistic Analysis Using LIDA. In *Proceedings of 23rd Intl. Conf. on Software Engineering (ICSE 2001)*, Toronto, Canada

(Pohl et al., 2001) Pohl, K., Brandenburg, M., and Gülich, A. Integrating requirement and architecture information: A scenario and meta-model based approach. In *Proc. of the 7th Intl. Workshop on Requirements Engineering: Foundation for Software Quality* (REFSQ'01). Interlaken, Switzerland, 2001.

(Pohl, 1993) Pohl, K. The three dimensions of requirements engineering. In Rolland, C., Bodart, F., and Cauvet, C. (Eds.) *Proc. of 5th Intl. Conf. on Advanced Information Systems Engineering* (CAiSE'93), Springer-Verlag, LNCS 685, Paris, France, 1993, pages 275-292.

(Poole & Warner, 2000) Poole, M., and Warner, M. *The IEBM Handbook of Human Resource Management*. London: Thomson learning, 2000. 945 p.

(PorstgreSQL, 2005) PostgreSQL: The world's most advanced open source database. http://www.postgresql.org/. (Last checked: 2005 10 03)

(Preece et al., 1994) Preece, J., Rogers, Y., Sharp, H., Benyon, D., Holland, S., and Carey, T. *Human-Computer Interaction*. Addison-Wesley Publishing, 1994.

(Proper & Hoppenbrouwers, 2004) Proper, H.A., and Hoppenbrouwers, S.J.B.A. Concept evolution in information system evolution. In Gravis, J., Persson, A., and Stirna, J. (Eds.), *Forum proc. of the 16th Conf. on Advanced Information Systems 2004* (CAiSE 2004), Riga Technical University, Riga, Latvia, 2004, ISBN 998497670X, pages 63-72.

(Ramazani et al., 1998) Ramazani, D., Bochmann, G.V., and Flocchini, P. Object Naming and Object Composition. Publication #1135, Université de Montréal, Canada, 1998.

(Ramesh & Jarke, 2001) Ramesh, B., and Jarke, M. Toward reference models for requirements traceability. *IEEE Transactions on Software Engineering* 27(1), January 2001, pages 58–93.

(Reduce, 2005) REDUCE Home Page. http://www.cit.gu.edu.au/~scz/projects/reduce/. (Last checked: 2005 12 28).

(Reinhartz-Berger et al., 2005) Reinhartz-Berger, I., Sturm, A., and Wand, Y. Tutorial 3: Domain Engineering - Using Domain Concepts to Guide Software Design. In Akoka, J. et al. (Eds.) *Perspectives in Conceptual Modeling: ER 2005 Workshops*, LNCS 3770, Springer-Verlag, 2005, pages 461-463.

(Riemenschneider et al., 2002) Riemenschneider, C.K., Hardgrave, B.C., and Davis, F.D. Explaining Software Developer Acceptance of Methodologies: A Comparison of Five Theoretical Models. *IEEE Transactions on Software Engineering* 28(12), 2002, pages 1135-1145.

(RM-ODP, 2005) RM-ODP: The Reference Model for Open Distributed Processing. URL: http://www.rm-odp.net/. Last checked: (2005 12 28).

(Rolland & Prakash, 2000) Rolland, C., and Prakash, N. From conceptual modelling to requirements engineering. *Annals of Software Engineering* 10, 2000, pages 151-176.

(Rolland & Proix, 1992) Rolland, C., and Proix, C. A Natural Language Approach for Requirements Engineering. In Loucopoulos, P. (Ed.) *Proc. of 4th Intl. Conf. on Advanced Information Systems Engineering* (CAiSE'92), LNCS 593, Springer-Verlag, 1992, pages 257-277.

(Roschelle & Teasley, 1995) Roschelle, J., and Teasley, S. The construction of shared knowledge in collaborative problem solving. In O'Malley, C.E. (Ed.), *Computer Supported Collaborative Learning*, 1996, Springer-Verlag, pages 69-97.

(Royce, 1987) Royce, W. Managing the development of large software systems. In Thayer, R.H. (Ed.) *Tutorial: Software Engineering Project Management*. IEEE Computer Society, Washington, 1987, pages 118–127.

(Saeki, 2004) Saeki, M. Ontology-Based Software Development Techniques. *ERCIM News*, No. 58, July 2004, p. 14. URL: http://www.ercim.org/publication/ Ercim_News/enw58/saeki.html. (Last checked: 2005 09 28)

(Sandpiper Software, 2005) Sandpiper Software. Medius Visual Ontolgoy Modeler. URL: http://www.sandsoft.com/products.html. (Last checked: 2005 10 03)

(Schmidt & Bannon, 1992) Schmidt, K. and Bannon, L. Taking CSCW seriously. *CSCW, 1992* Vol. 1 (No.1-2), pages 7-40.

(Schutte & Rotthowe, 1998) Schütte, R., Rotthowe, T.: The guidelines of modeling - an approach to enhance the quality in information models. In Ling, T.W., Ram, S., Lee, M.L. (Eds.) *Proc. of 17th Intl. Conf. on Conceptual Modeling* (ER'98). LNCS 1507, Springer-Verlag, 1998, pages 240-254.

(SEI, 1994) Software Engineering Institute. Software Configuration Management. CMU, 1988-1994. URL: http://www.sei.cmu.edu/legacy/scm/ (Last checked: 2005 09 29)

(Seltveit, 1994) Seltveit, A. *Complexity Reduction in Information Systems Modelling*. PhD thesis, NTH, Trondheim, Norway, 1994.

(SEPT, 2005) Software Engineering Process Technology. Information on ISO/IEC 12207 and other software engineering standards. URL: http://www.12207.com/. (Last checked: 2005 11 20)

(Shen & Sun, 2002) Shen, H., and Sun, C. Flexible Notification for Collaborative Systems. In *Proc. of CSCW'02,* 2002, New Orleans, Louisiana, USA. 2002, pages 77-86.

(Siegel & Castellan, 1988) Siegel, S., and Castellan, N.J. *Nonparametric statistics for the behavioural sciences*. 2nd edition, McGraw-Hill, 1988.

(Sindre, 1990) Sindre, G. *HICONS: A General Diagrammatic Framework for Hierarchical Modelling*. PhD thesis, NTH, Trondheim, Norway, 1990.

(Sobalipse, 2005) Sobalipse plugin. http://sobalipse.sourceforge.net/. (Last checked: 2005 12 28)/

(Solvberg, 2000) Sølvberg, A. Co-operative Concept Modeling. In Brinkkemper, S., Lindencrona, E. and Sølvberg, A. (Eds.), *Information Systems Engineering – State of the Art and Research Themes*, Springer-Verlag, Berlin, 2000, pages 305–317.

(Solvberg, 1999) Sølvberg, A. Data and what they refer to. In Chen, P., Akoka, J., Kangassalo, H., Thalheim, B. (Eds.), *Conceptual Modeling: Current Issues and Future Trends*. LNCS 1565. Springer-Verlag, 1999, pages 211-226.

(Solvberg et al., 2002) Sølvberg, A., Hakkarainen, S., Brasethvik, T., Su, X., Matskin, M., and Strasunskas, D. Concepts of Enriching, Understanding and Retrieving the Semantics on the Web. *ERCIM News*, No. 51, October 2002, pages 41-42. URL: http://www.ercim.org/publication/Ercim_News/enw51/solvberg.html (Last checked: 2005 09 12)

(Solvberg & Brasethvik, 2000) Sølvberg, A., and Brasethvik, T. *The referent model language*. Technical report, NTNU, Trondheim, Norway URL: http://www.idi.ntnu.no/~ppp/referent/. (Last checked: 2005 09 28)

(Solvberg & Kung, 1993) Sølvberg, A., and Kung, D.C. *Information Systems Engineering – An Introduction*, Springer-Verlag, 1993.

(Sommerville, 1992) Sommerville, I. *Software engineering*. 4th edition, Addison-Wesley: Wokingham, England, 1992, 649 p.

(Spanoudakis et al., 2004) Spanoudakis, G., Zisman, A., Pérez-Miñana, E., and Krause, P. Rule-based Generation of Requirements Traceability Relations. *Journal of Systems and Software* 72(2), 2004, pages 105-127.

(StandishGroup, 1994) The Standish Group, Inc. The CHAOS Report, 1994. URL: http://www.standishgroup.com/sample_research/chaos_1994_2.php. (Last checked: 2005 09 29)

(St-Denis et al., 2000) St-Denis, G., Schauer, R., and Keller, R.K. Selecting a Model Interchange Format The SPOOL Case Study. In *IEEE Proc. of the 33rd Annual Hawaii Intl. Conf. on System Sciences*, Maui, Hawaii, 2000.

(Steele, 2004) Steele, O. An OO interface to the WordNet database. URL: http://sourceforge.net/projects/pywordnet. (Last checked: 2005 08 20)

(Strasunskas, 2003) Strašunskas, D. A Vision for Product Traceability based on Semantics of Artifacts. In Al-Ani, B., Arabnia, H.R., and Mun, Y. (Eds.) *Proc. of the 2003 Intl. Conf. on Software Engineering Research and Practice* (SERP'2003), part of Intl. MultiConference in Computer Science & Engineering, CSREA Press, Vol.II, ISBN:1-932415-20-3, Las Vegas, Nevada, USA, June 2003, pages 890-895.

(Strasunskas, 2002) Strašunskas D. Traceability in Collaborative Systems Development from Lifecycle Perspective - a position paper. In *Proc. of the 1st Intl. Workshop on Traceability*, co-located with ASE 2002, Edinburgh, Scotland, UK, September 2002, pages 54-60.

(Strasunskas et al., 2006) Strašunskas, D., Lin, Y., and Hakkarainen, S. Domain knowledge-based reconciliation of model fragments. In A.G. Nilsson et al. (Eds.), *Advances in Information Systems Development: Bridging the Gap between Academia and Industry*, Springer, 2006, *in press*.

(Strasunskas et al., 2004) Strašunskas, D., Fidjestøl, A.D., Hakkarainen, S., Lin, Y., and Sølvberg, A. Repository design. Technical Report, IDI, Norwegian University of Science and Technology (NTNU), Trondheim, Norway, 2004.

(Strasunskas et al., 2003) Strašunskas, D., Fidjestøl, A.D., and Hakkarainen, S. Product fragment repository – Requirements specification. Technical Report, IDI, Norwegian University of Science and Technology (NTNU), Trondheim, Norway, 2003.

(Strasunskas & Hakkarainen, 2004) Strašunskas, D., and Hakkarainen, S. Domain Model Driven Approach to Change Impact Assessment. In Linger, H. et al. (Eds.), *Constructing the Infrastructure for the Knowledge Economy: Methods and Tools, Theory and Practice*, Kluwer Academic / Plenum Publishers, 2004, pages 305-316.

(Strasunskas & Hakkarainen, 2003) Strašunskas, D., and Hakkarainen, S. Process of Product Fragments Management in Distributed Development. In Meersman, R., Tari, Z., Schmidt, D. et al. (Eds.) *Proc. of the 11th Intl. Conf. on Cooperative Information Systems* (CoopIS'2003), Springer-Verlag, LNCS 2888, Catania, Sicily, Italy, November 2003, pages 218-234.

(Strasunskas & Lin, 2005) Strašunskas, D., and Lin, Y. Model and knowledge management in distributed development: agreement based approach. In Vasilecas, O. et al. (Eds.), *Information Systems Development: Advances in Theory, Practice, and Education*, Springer, 2005, pages 389-402.

(Su, 2004) Su, X. *Semantic Enrichment for Ontology Mapping*. PhD thesis, Norwegian University of Science and Technology (NTNU), Trondheim, Norway, 2004.

(Sun, 2002) Sun, C. Undo as concurrent inverse in group editors. *ACM Transactions on Computer-Human Interaction* 9(4), December 2002, pages 309-361.

(Sun et al., 1998) Sun, C., Jia, X., Zhang, Y., Yang, Y., and Chen, D. Achieving convergence, causality-preservation, and intention-preservation in real-time cooperative editing systems. *ACM Transactions on Computer-Human Interaction* 5(1), March, 1998, pages 63-108.

(Sutcliffe & Maiden, 1998) Sutcliffe, A., and Maiden, N.A.M. The domain theory for requirements engineering. *IEEE Transactions On Software Engineering* 24(3), 1998, pages 174-196.

(Suzuki & Yamamoto, 1998) Suzuki, J., and Yamamoto, Y. Managing the Software Design Documents with XML. In *Proceedings of the Sixteenth Annual International Conference of Computer Documentation* (ACM SIGDOC '98), Quebec City, Canada, 1998, pages 127-136.

(Telelogic, 2005) Telelogic Doors. URL: http://www.telelogic.com/corp/products/doors/doors/index.cfm. Last checked: (2005 12 28).

(Toranzo & Castro, 1999) Toranzo, M.A., and Castro, J.F.B. A Comprehensive Traceability Model to Support the Design of Interactive Systems. In *Proc. of Intl. Workshop on Interactive system Development and Object Models* (WISDOM99), Lisboa, Springer-Verlag, LNCS 1743, 1999, pages 283-284.

(Traetteberg, 2002) Trætteberg, H. *Model-based User Interface Design*. PhD thesis, Norwegian University of Science and Technology (NTNU), Trondheim, Norway, 2002.

(UDDI, 2005) Universal Description, Discovery and Integration. URL: http://www.uddi.org/ (Last checked: 2005 12 12)

(Unicorn, 2005) Unicorn. Unicorn Workbench. http://unicorn.com/products/unicornsystem/workbench.htm. (Last checked: 2005 12 28).

(VA Software, 2004) VA Software. Leveraging Open Source Processes and Techniques in the Enterprise. White Paper. VA Software, November, 2004.

(Vlist, 2003) van der Vlist, E. *RELAX NG*. O'Reilly, December 2003. URL: http://books.xmlschemata.org/relaxng/. (Last checked: 2005 11 20)

(Visser et al., 1998) Visser, P., Jones, D.M., Bench-Capon, T., and Shave, M. Assessing heterogeneity by classifying ontology mismatches. In *Proc. of the Intl. Conf. on Formal Ontology in Information Systems* (FOIS'98), Trento, Italy, 1998, pages 148-162.

(W3C, 2005) W3C. OWL- Web Ontology Language Overview. http://www.w3.org/TR/owl-features/. (Last checked: 2005 12 20)

(Wache et al., 2001) Wache, H., Vögele, T., Visser, U., Stuckenschmidt, H., Schuster, G., Neumann, H., and Hübner, S. Ontology-Based Integration of Information A Survey of Existing Approaches. In H. Stuckenschmidt (Ed.) *Proc. of IJCAI-01 Workshop: Ontologies and Information Sharing*, 2001, pages 108-117.

(Weibel et al., 1998) Weibel, S., Kunze, J., Lagoze, C. and Wolf, M. Dublin core metadata for resource discovery, Technical Report RFC2413, Internet Engineering Task Force (IETF), 1998. URL: http://ww.ietf.org/rfc/rfc2413.txt. (Last checked: 2005 08 29)

(Wiederhold, 1994) Wiederhold, G. An Algebra for Ontology Composition. In *Proc. of 1994 Monterey Workshop on Formal Methods*, U.S. Naval Postgraduate School, 1994, pages 56-61.

(Wieringa & de Jonge, 1995) Wieringa, R., and de Jonge, W. Object identifiers, keys, and surrogates – object identifiers revisited. *Theory and Practice of Object Systems* 1(2), 1995, pages 101-114.

(Wikipedia/Cooperation, 2005) Wikipedia/Co-operation. URL: http://en.wikipedia.org/wiki/Co-operation. (Last checked: 2005 08 22)

(Wikipedia/Collaboration, 2005) Wikipedia/Collaboration. URL: http://en.wikipedia.org/wiki/Collabora-tion. (Last checked: 2005 08 22)

(Willumsen, 1993) Willumsen, G. *Executable Conceptual Models in Information Systems Engineering*. PhD thesis, NTH, Trondheim, Norway, 1993.

(Wilson, 1991) Wilson, P. *Computer Supported Cooperative Work: An introduction*. Intellect Books, Oxford, UK, 1991.

(Wu & Graham, 2004) Wu, J., and Graham, T.C.N. The Software Design Board: a Tool Supporting Workstyle Transitions in Collaborative Software Design. In Bastide, R., Palanque, P., and Roth, J. (Eds.), *Proc. of EHCI-DSVIS 2004*, Springer-Verlag, LNCS 3425, 2005, pages 363-382.

(Wu & Palmer, 1994) Wu, Z., and Palmer, N. Verbs Semantics and Lexical Selection. In *Proc. of the 32nd Conf. on Association for Computational Linguistics*, New Mexico, Association of Computational Linguistics, 1994, pages 133-138.

(Xia et al., 2004) Xia, S., Sun, D., Sun, C., Chen, D., and Shen, H. Leveraging single-user applications for multi-user collaboration: the CoWord approach. In *Proc. of ACM 2004 Conf. on Computer Supported Cooperative Work,* Chicago, IL USA, pages 162-171.

(Zachman, 1987) Zachman, J.A. A framework for information systems architecture. *IBM Systems Journal* 26(3), 1987, pages 276-292.

(Zhdanova et al., 2005) Zhdanova, A.V., Krummenacher, R., Henke, J., and Fensel, D. Community-Driven Ontology Management: DERI Case Study. In Skowron, A. et al. (Eds.) *Proc. of 2005 IEEE/WIC/ACM Intl. Conf. on Web Intelligence*. IEEE Computer Society Press, September 2005, pages 73-79.

Appendix A
The Referent Model Language

The referent model language has been developed in the Information Systems group at IDI, NTNU. The RML language is a modelling language that initially springs out from the PPP integrated modelling environment (Gulla et al., 1991). PPP initially contained support for several modelling languages; a process model language PrM, an extended ER modelling language (ONE-R) and a rule modelling language (PLD).

Later work have refined the initial modelling languages and also added new languages. The most recent are the RML concept modelling language (Solvberg, 1999), the APM workflow modelling language (Carlsen, 1997), and the task modelling and dialogue modelling languages for user interface design (Traetteberg, 2002).

A.1 RML Foundation

The Referent Model Language (RML) is a concept modelling language targeted towards applications in areas of information management and heterogeneous organisation of data (Solvberg, 1999). It has a formal basis from set theory and provides a simple and compact graphical modelling notation (see Figure A.1) for set theoretic definitions of concepts and their relations.

A.2 Basic Concepts

In RML, semantics of concepts are defined through set theoretic constructs such as intension, extension and reference. RML defines constructs for modelling of concepts, the selection of constructs is based on the concept types given by (Bunge, 1998):

Individual concepts - individual concepts apply to individuals. Individuals can be either specific or generic;

Class concepts - concepts that apply to collections of individuals;

Relation concepts - concepts that refer to relations among objects (individual or class concepts). Distinction between class concepts and relation concepts is vague, as a relation may be considered a class concept in its own right.

Quantitative concepts - quantitative concepts do not represent distinct objects, but refer to magnitudes often associated with individual or class concepts.

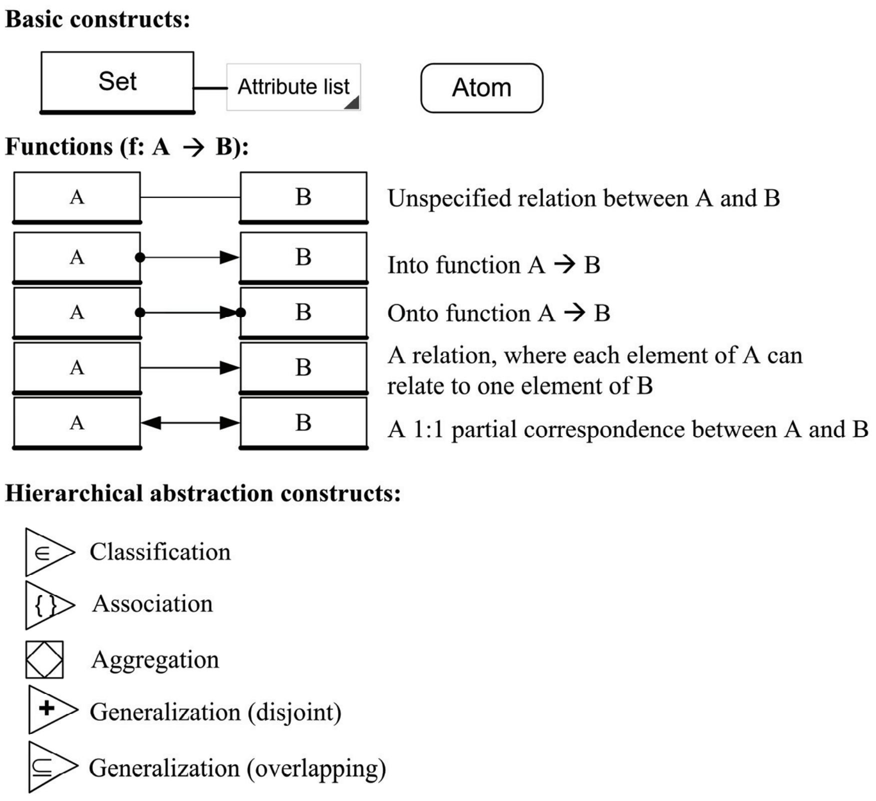

Basic constructs:
Set
Attribute list
Atom
Functions (f: A → B):
A
B
Unspecified relation between A and B
Into function A → B
Onto function A → B
A relation, where each element of A can relate to one element of B
A 1:1 partial correspondence between A and B
Hierarchical abstraction constructs:
Classification
Association
Aggregation
Generalization (disjoint)
Generalization (overlapping)

Figure A.1 Graphical notation of RML

A.3 RML Meta-Model

RML meta-model is denoted in Figure A.2 using RML graphical notation introduced in above. The intension of a concept is the set of all characteristic properties of the concept. A characteristic property of a concept is a property that is shared by all the referents of a concept. The attributes of a concept are defined as a list of properties and are drawn in a rectangle with a small black triangle in its lower right corner.

The referent is what the concept refers to. The referent set of a concept contains all members; past, present and future, imaginary or real. The extension of a concept is all individuals that belong to the concept.

Two kinds of constraints may be applied to a relation: cardinality and coverage. The cardinality of a relationship is defined as the number of members from each of the corresponding sets that participates in the relation. The cardinality of a relation is shown with the use of an arrow or by specifically numbering the maximum number of participating members from the set.

Relations may be given names. Names are written on top of the relation, the arrow above indicates the direction of reading the relation name.

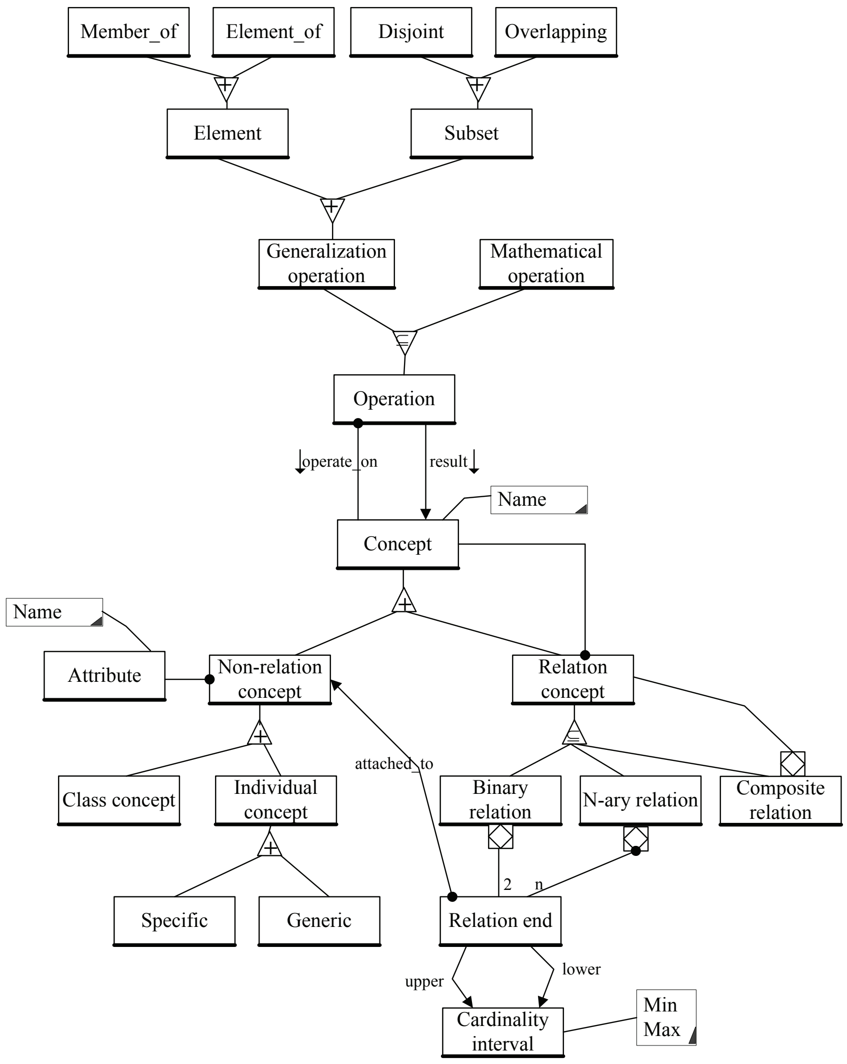


Figure A.2 RML meta-model

RML supports several abstraction constructs of semantic modelling. All of them have set theoretical counterparts:

- Classification: specific instances are considered as a higher level object type via the *is instance of* relationship.
- Aggregation: an object is related to the components that make it up via the *is part of* relationship.
- Generalisation: similar object types are abstracted into a higher level object type via the *is-a* relationship.
- Association: several object types are considered a higher level set object type via the *is a member of* relationship.

N-ary relations are modelled as composite class concepts. Composing a concept of other concepts is performed by using regular relations as part-of relations from the part class concepts to the composite class concept.

As relations can be considered concepts in their own right, hence also relations can be composed. In RML a composition of relations is defined as a derived relation. From a set theoretical perspective, derived relations correspond to composition of functions. However, from a pure modelling perspective, they can also be viewed as a simple naming of a path in the model, i.e. a kind of short-cut or shorthand notation.

A.4 RelaxNG RML Schema

RML model is saved as an XML file. Semantic information of the model is stored separately in the file. Figure A.3 shows Relax NG[22] (REgular LAnguage for XML Next Generation) schema (Vlist, 2003) for RML model.

```
<grammar>
  <start>
   <element name="referent-diagram" xmlns="http://relaxng.org/ns/structure/1.0">
     <zeroOrMore>
       <element name="referent">
         <attribute name="id"/>
         <zeroOrMore>
         <ref name="name"/>
         <element name="aggregation">
           <attribute name="id"/>
           <text/>
         </element>
         <element name="attribute">
           <attribute name="attribute"/>
           <text/>
         </element>
         </zeroOrMore>
       <element name="operation">
         <attribute name="id"/>
         <attribute name="operation type">
           <choice>
             <value>isa</value>
             <value>subset</value>
             <value>member-of</value>
             <value>element-of</value>
             <value>disjoint</value>
           </choice>
         </attribute>
         <attribute name="operation direction">
           <choice>
             <value>up</value>
             <value>down</value>
```

[22] Relax NG is a schema language for XML. A Relax NG schema specifies a pattern for the structure and content of an XML document. A Relax NG schema is itself an XML document. Furthermore, Relax NG also offers a popular compact, non-XML syntax. Compared to other popular schema languages, Relax NG is relatively simple.

```
                        <value>right</value>
                        <value>left</value>
                    </choice>
                </attribute>
                <text/>
            </element>
            <element name="dataset">
                <ref name="name"/>
                <attribute name="id"/>
                <text/>
            </element>
            <element name="element">
                <ref name="name"/>
                <attribute name="id"/>
                <text/>
            </element>
            <element name="relation">
                <twoOrMore>
                    <element name="relation-end">
                        <attribute name="idref"/>
                        <attribute name="cardinality">
                            <choice>
                                <value>one</value>
                                <value>many</value>
                            </choice>
                        </attribute>
                        <attribute name="coverage">
                            <choice>
                                <value>partial</value>
                                <value>full</value>
                            </choice>
                        </attribute>
                    </element>
                    <ref name="name"/>
                </twoOrMore>
                <text/>
            </element>
            <element name="operation-link">
                <element name="link-from">
                    <attribute name="idref"/>
                </element>
                <element name="link-to">
                    <attribute name="idref"/>
                </element>
                <text/>
            </element>
            <element name="canvas-text">
                <ref name="name"/>
                <text/>
            </element>
        </zeroOrMore>
   </element>
  </start>
  <define name="name">
   <element name="text">
      <text/>
   </element>
```

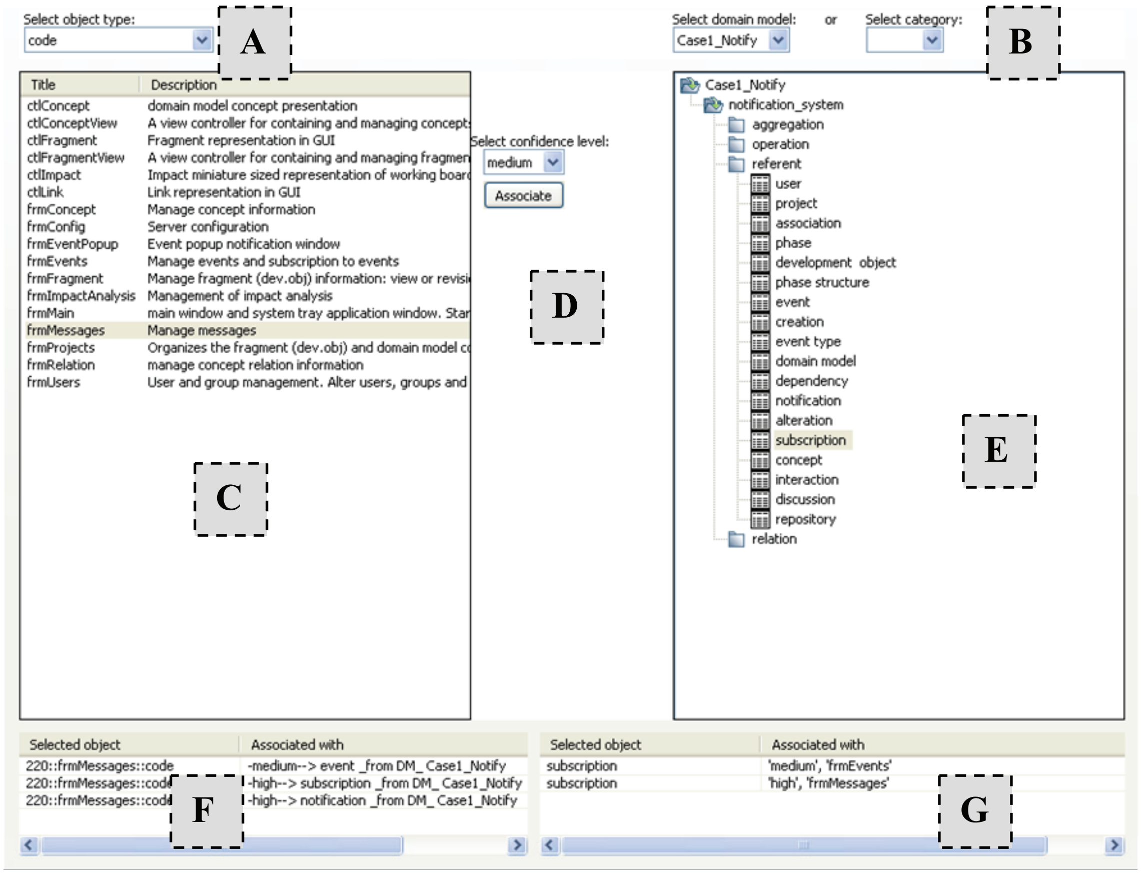
```
    </define>
</grammar>
```

Figure A.3 RML schema in RelaxNG

Appendix B
Prototype visualisation

Here we provide an overview of graphical user interface and functionality implemented in a prototype system CO_2SY. This is an additional material for chapter 7.

The main functionality of the implemented prototype is visualized by illustrating "*B area*" (recall Figure 7.6) of particular functionality. Figure B.1 shows the screenshot of window for product fragment association with concepts from domain model. The main components of this window are as follows.



Select object type: code

Select domain model: Case1_Notify

Select confidence level: medium

Title	Description
ctlConcept	domain model concept presentation
ctlConceptView	A view controller for containing and managing concept
ctlFragment	Fragment representation in GUI
ctlFragmentView	A view controller for containing and managing fragmen
ctlImpact	Impact miniature sized representation of working boar
ctlLink	Link representation in GUI
frmConcept	Manage concept information
frmConfig	Server configuration
frmEventPopup	Event popup notification window
frmEvents	Manage events and subscription to events
frmFragment	Manage fragment (dev.obj) information: view or revisi
frmImpactAnalysis	Management of impact analysis
frmMain	main window and system tray application window. Star
frmMessages	Manage messages
frmProjects	Organizes the fragment (dev.obj) and domain model c
frmRelation	manage concept relation information
frmUsers	User and group management. Alter users, groups and

- Case1_Notify
  - notification_system
    - aggregation
    - operation
    - referent
      - user
      - project
      - association
      - phase
      - development object
      - phase structure
      - event
      - creation
      - event type
      - domain model
      - dependency
      - notification
      - alteration
      - subscription
      - concept
      - interaction
      - discussion
      - repository
    - relation

Selected object	Associated with
220::frmMessages::code	-medium--> event _from DM_ Case1_Notify
220::frmMessages::code	-high--> subscription _from DM_ Case1_Notify
220::frmMessages::code	-high--> notification _from DM_ Case1_Notify

Selected object	Associated with
subscription	'medium', 'frmEvents'
subscription	'high', 'frmMessages'

Figure B.1 Association with a concept (classification) interface

A – selection of an object type;

B – selection of a domain model or category (e.g., classifying the code fragments according model – view – controller paradigm).

C – a list of selected type of development objects;

D – selection of confidence level, only when associating with domain model;

E – a list of selected domain model elements (or category). In current version only a simple list of model elements is shown. Future improvements (after tight integration with modelling environment) will have normal graphical model view;

F – a list of all associations for a particular selected development object;

G – a list of all development objects of a certain type associated with a selected concept.

Figure B.2 visualizes the main window for the repository content browsing. The main components of this window are as follows.

A – selection of an object type;

B – a list of selected type of development objects;

C – meta data. Here description can be provided in natural language or by URL. Language of the development object is specified, depending on a particular type of development object;

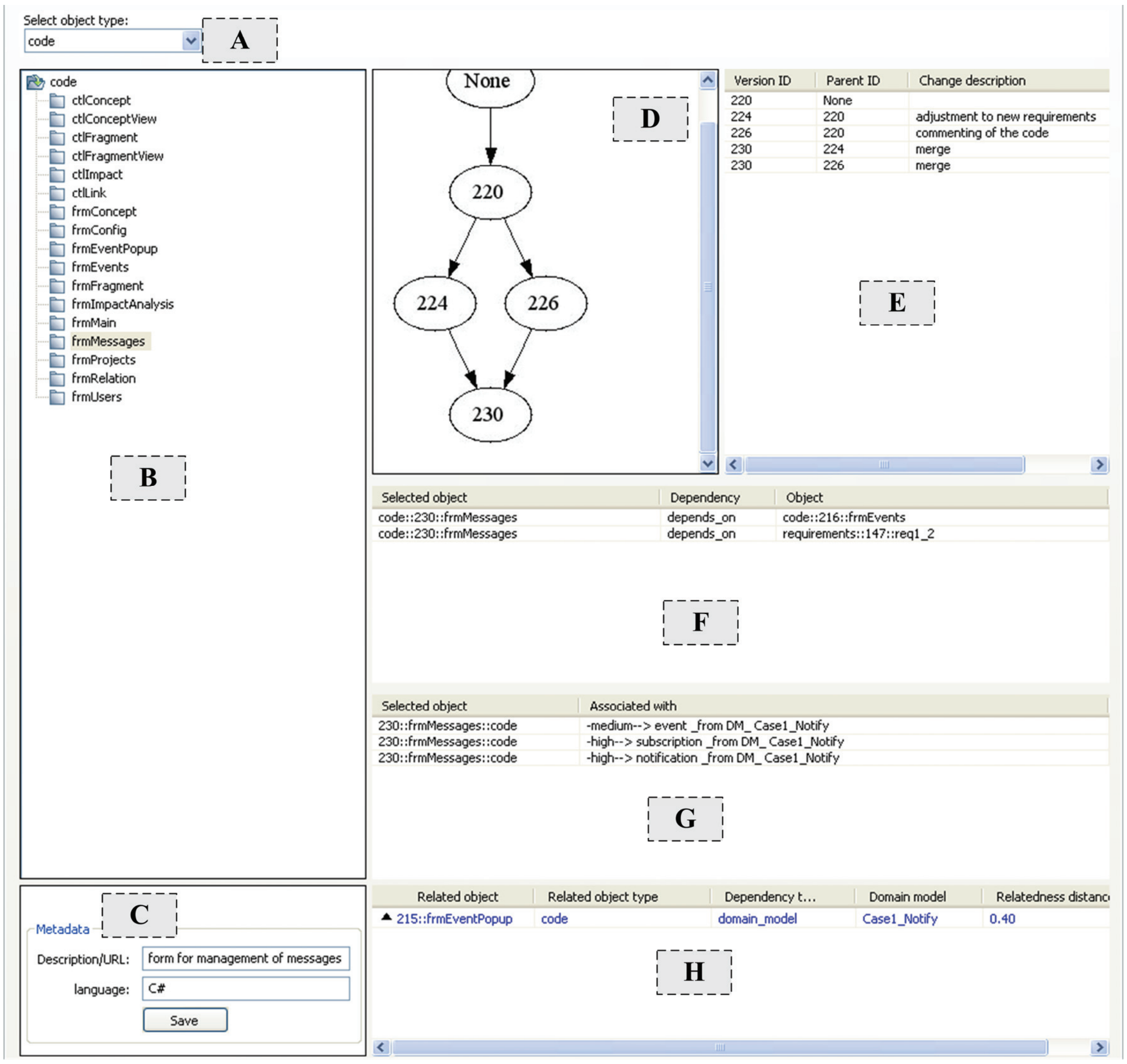
D – a version graph;

E – a version list, including parent and change description;

F – a list of direct dependencies for the selected product fragment;

G – a list of associations with concepts;

H – a list of possible impacts for the selected development object.



Select object type: code

- code
  - ctlConcept
  - ctlConceptView
  - ctlFragment
  - ctlFragmentView
  - ctlImpact
  - ctlLink
  - frmConcept
  - frmConfig
  - frmEventPopup
  - frmEvents
  - frmFragment
  - frmImpactAnalysis
  - frmMain
  - frmMessages
  - frmProjects
  - frmRelation
  - frmUsers

Metadata

Description/URL: form for management of messages

language: C#

Version ID	Parent ID	Change description
220	None	
224	220	adjustment to new requirements
226	220	commenting of the code
230	224	merge
230	226	merge

Selected object	Dependency	Object
code::230::frmMessages	depends_on	code::216::frmEvents
code::230::frmMessages	depends_on	requirements::147::req1_2

Selected object	Associated with
230::frmMessages::code	-medium--> event _from DM_ Case1_Notify
230::frmMessages::code	-high--> subscription _from DM_ Case1_Notify
230::frmMessages::code	-high--> notification _from DM_ Case1_Notify

Related object	Related object type	Dependency t...	Domain model	Relatedness distanc
▲ 215::frmEventPopup	code	domain_model	Case1_Notify	0.40

Figure B.2 Content browsing. "Rich" information about a development object

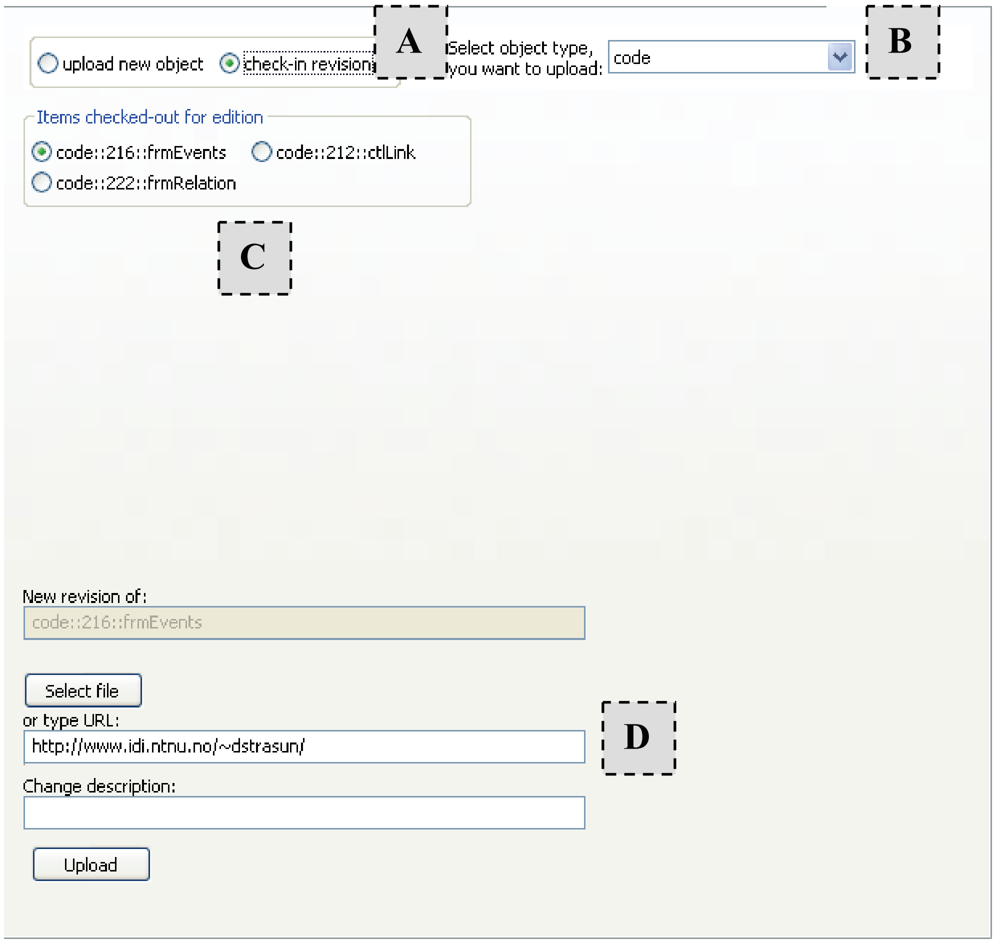
Figure B.3 illustrates the form used to upload a new development object or a new revision of a development object. The main components of this form are as follows.

A – a radio box field, allowing to select whether a new development object or a new revision of development object will be uploaded.

B – selection of development object type

C – a list of development objects checked-out for revision for a particular user.

D – file selection, change description window. When uploading the new development object only D window is displayed.



Select object type, you want to upload: code

Items checked-out for edition

[✓ code::216::frmEvents] code::212::ctlLink

code::222::frmRelation

New revision of: code::216::frmEvents

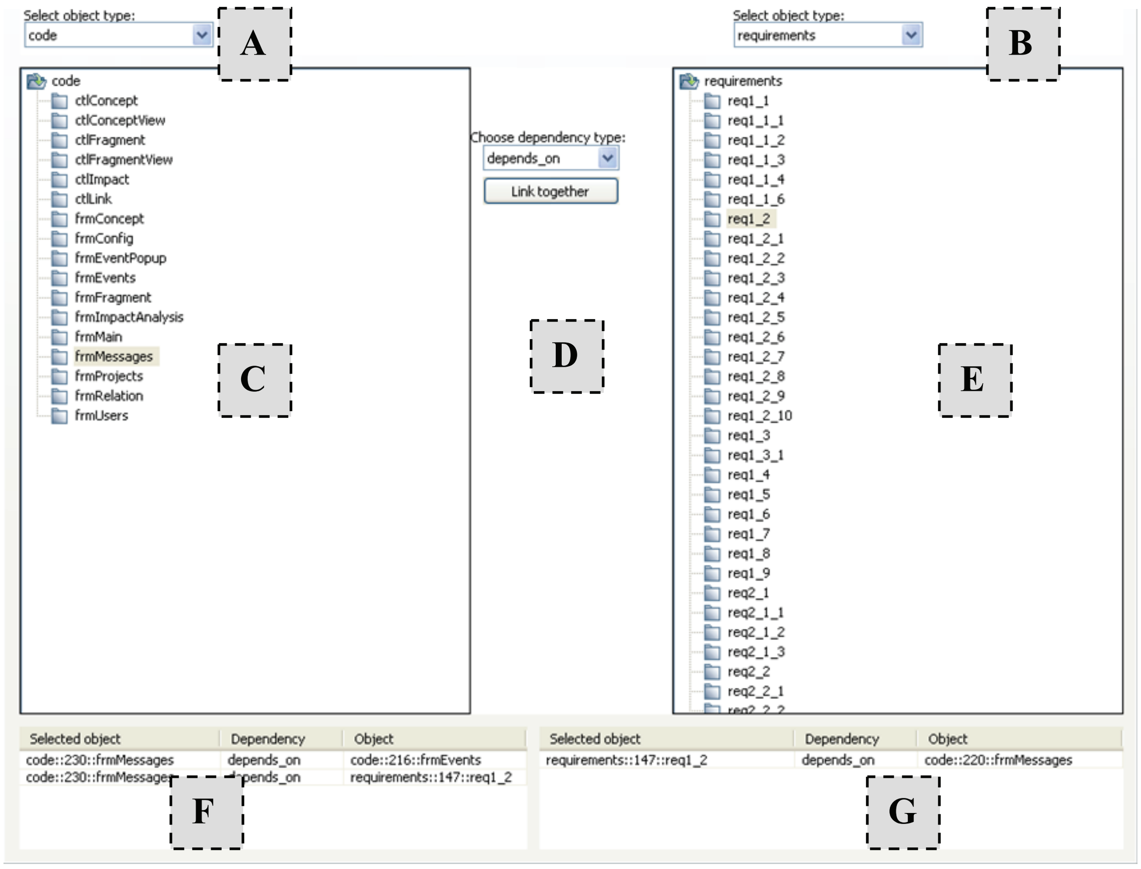
or type URL: http://www.idi.ntnu.no/~dstrasun/

Change description:

Figure B.3 New object / revision upload interface

Figure B.4 similar as Figure B.1, just right part of the window is for development object type selection instead of domain model.

Figure B.5 illustrates the screenshot of repository querying form. The form is made using XRC (XML Resources) forms, i.e. XML based form definition. This flexibility is necessary to maintain and adopt the query form for new object types. The form is made based on the searched object type, i.e. whether the selected object is binary or not. For binary objects only simple search using the name of development object and metadata is available. For non-binary development objects the query form structure is generated from the entries in table *development object_elements* see Figure 7.4.



Select object type: code

code
- ctlConcept
- ctlConceptView
- ctlFragment
- ctlFragmentView
- ctlImpact
- ctlLink
- frmConcept
- frmConfig
- frmEventPopup
- frmEvents
- frmFragment
- frmImpactAnalysis
- frmMain
- frmMessages
- frmProjects
- frmRelation
- frmUsers

Choose dependency type: depends_on

Link together

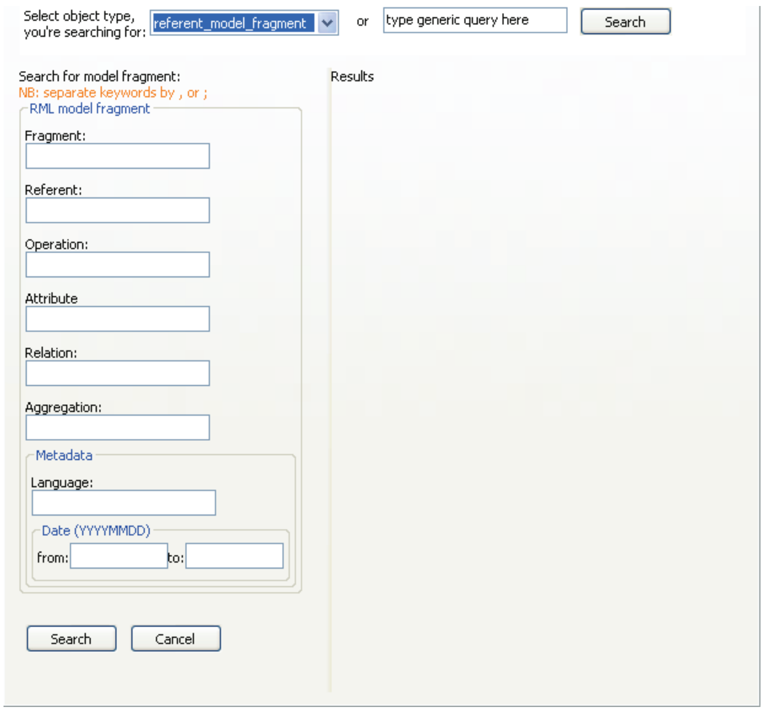
Select object type: requirements

requirements
- req1_1
- req1_1_1
- req1_1_2
- req1_1_3
- req1_1_4
- req1_1_6
- req1_2
- req1_2_1
- req1_2_2
- req1_2_3
- req1_2_4
- req1_2_5
- req1_2_6
- req1_2_7
- req1_2_8
- req1_2_9
- req1_2_10
- req1_3
- req1_3_1
- req1_4
- req1_5
- req1_6
- req1_7
- req1_8
- req1_9
- req2_1
- req2_1_1
- req2_1_2
- req2_1_3
- req2_2
- req2_2_1
- req2_2_2

Selected object	Dependency	Object
code::230::frmMessages	depends_on	code::216::frmEvents
code::230::frmMessages	depends_on	requirements::147::req1_2

Selected object	Dependency	Object
requirements::147::req1_2	depends_on	code::220::frmMessages

Figure B.4 Direct dependency linking



Select object type, you're searching for: referent_model_fragment or type generic query here Search

Search for model fragment:
NB: separate keywords by , or ;

RML model fragment
Fragment:
Referent:
Operation:
Attribute
Relation:
Aggregation:

Metadata
Language:

Date (YYYYMMDD)
from: to:

Search Cancel

Results

Figure B.5 Query interface

Appendix C
Questionnaire

This appendix presents a questionnaire used in the experiment described in chapter 8.

Dear participant,

I invite you to take part in an experiment to evaluate tools/ techniques for management of relatedness and dependency of development objects (e.g., product fragments) in a context of software systems development. The goal of the research is to validate a method for managing product fragments relatedness, change impact, and traceability in systems development. Therefore, I would like to test usability of the implemented prototype system (named, CO_2SY), mainly focusing on an applicability of the approach, not a tool, and to compare with other de facto standards in the area. The experiment is focused on various aspects of dependency (relatedness) links establishment and management.

This questionnaire is designed to discover what features (functionality) are most essential in dependency management, compare and evaluate proposed method and its implementation. The results of the questionnaire will be used to improve the method and prototype tool. All the collected data will be highly confidential and will be used only for study and research purposes. Your input is valuable and is of great importance in helping me to create successful and useful method. In particular I have no intention of judging you – I am merely interested in collecting data about above defined tools usage and usability.

Thanks in advance for the cooperation!

Questionnaire

Background Data

Q1. Gender	☐ *Male* ☐ *Female*

Q2. Age	☐ *21-30 years;*
	☐ *31-40 years;*
	☐ *41-50 years;*
	☐ *above 50.*

Q3. Have you ever been involved in an industrial software development project? Check all that apply.	☐ *No, I've never done anything like that;* ☐ *I've been observing (auditing, consulting, researching) a software development project;* ☐ *I've participated in a small software development project* ☐ *I've participated in a big software development project* ☐ *I've participated in a big geographically distributed software development project.*
Q4. Have you ever been involved in usability evaluations? Check all that apply.	☐ *No, I've never done anything like that before* ☐ *I've answered questionnaires or surveys pertaining to usability* ☐ *I've contributed to user testing as a participant* ☐ *I've conducted evaluations* ☐ *I've conducted user testing*
Q5. How long you are (working) in the field of computer and information science	☐ *less than 1 year* ☐ *1-5 years* ☐ *6-10 years* ☐ *11-25 years* ☐ *over 25 years*

Evaluation Tasks and Questions

Q6. Please select random 3 (from marked ones) development object of any type in CO$_2$SY, select 3 top ranked values (lowest value) and check how many of them are correctly estimated as being related/dependent. Please do that for each of the cases. How many of these would you rate as correct?

Case 1:

Totally wrong	Partial	Totally correct		**Total # inspected**

Case 2:

Totally wrong	Partial	Totally correct		**Total # inspected**

Please provide justification or comment, if you wish:

Q7. Browse the results of association and linking (all tools), compare them. Please identify, whether CO$_2$SY help to discover any new correct relatedness/dependency relationship between development object.

Yes, but all were wrong	None	Neither, nor… just different way of achieving the same result	Some	Many
☐	☐	☐	☐	☐

Approach Evolution

Q8. What is your experience with:

Doors tool:

Never used	*Seen but not used*	*Used a few times*	*Used extensively*
☐	☐	☐	☐

CO$_2$SY tool:

Never used	*Seen but not used*	*Used a few times*	*Used extensively*
☐	☐	☐	☐

Traceability matrix:

Never used	*Seen but not used*	*Used a few times*	*Used extensively*
☐	☐	☐	☐

Q9. Please provide your opinion about the quality of case description:

Case 1

Very bad	Bad	Fair	Good	Very good
☐	☐	☐	☐	☐

Case 2

Very bad	Bad	Fair	Good	Very good
☐	☐	☐	☐	☐

quality of provided conceptual domain models:

Case 1

Very bad	Bad	Fair	Good	Very good
☐	☐	☐	☐	☐

Case 2

Very bad	Bad	Fair	Good	Very good
☐	☐	☐	☐	☐

and quality of the product fragment:

Case 1

Very bad	Bad	Fair	Good	Very good
☐	☐	☐	☐	☐

Case 2

Very bad	Bad	Fair	Good	Very good
☐	☐	☐	☐	☐

Q10. Please rank used tools / techniques based on perceived easiness and efficiency using it for this particular settings:

1 CO_2SY
1 Doors
1 Traceability matrix

Q11. Imagine a geographically distributed software development project bigger 10 times or more than the specified in the case 1. Please rank used tools / techniques based on perceived easiness and efficiency, but in the settings of big software development:

1 CO_2SY
1 Doors
1 Traceability matrix

Q12. When associating design and code fragments, did you experienced difficulty while choosing concept from domain model? If so, do you think it was because of (choose that apply):	☐ *No, I have not experienced any difficulty* ☐ *Lack of design/code oriented concepts* ☐ *Too abstract concepts in a domain model* ☐ *Lack of domain knowledge. If so, please specify:* ☐ *case 1,* ☐ *case 2* ☐ *I do not know the reason, e.g., the scale of an experiment was too limited, to say something trustworthy* ☐ *Other:*

Q13. In general how would you rate the user interface of:

Doors:

	Needs major improvements	Needs minor improvement	Fair	Good	Works well
Visual design / layout	☐	☐	☐	☐	☐
General ease of use	☐	☐	☐	☐	☐
Ease of learning	☐	☐	☐	☐	☐

CO_2SY:

	Needs major improvements	Needs minor improvement	Fair	Good	Works well
Visual design / layout	☐	☐	☐	☐	☐
General ease of use	☐	☐	☐	☐	☐
Ease of learning	☐	☐	☐	☐	☐

Q14. Are you satisfied with a linking functionality (for the purpose of identifying related/dependent items) of the tested tools/ techniques? Please use a scale from 1 to 5.

Doors:

Not at all				*Very satisfied*
1	2	3	4	5
☐	☐	☐	☐	☐

CO_2SY:

Not at all				*Very satisfied*
1	2	3	4	5
☐	☐	☐	☐	☐

Traceability matrix:

Not at all				*Very satisfied*
1	2	3	4	5
☐	☐	☐	☐	☐

Q15. Please evaluate the usability of the tools (usability considers how easy it is to learn and use a tool):

Doors tool:

Very difficult	Difficult	Moderate	Easy	Very easy
☐	☐	☐	☐	☐

CO_2SY tool:

Very difficult	Difficult	Moderate	Easy	Very easy
☐	☐	☐	☐	☐

Traceability matrix

Very difficult	Difficult	Moderate	Easy	Very easy
☐	☐	☐	☐	☐

Q16. Please evaluate easiness of dependency establishment using the tools (consider how labour intensive was the establishment of links/associations):

Doors tool:

Very difficult	Difficult	Moderate	Easy	Very easy
☐	☐	☐	☐	☐

CO_2SY tool:

Very difficult	Difficult	Moderate	Easy	Very easy
☐	☐	☐	☐	☐

Traceability matrix

Very difficult	Difficult	Moderate	Easy	Very easy
☐	☐	☐	☐	☐

Q17. Imagine a project relying on a well structured folder hierarchy in file system. The option would be to save file in corresponding folder, e.g., Project_name/User/Phase/… Please compare association with domain concept to saving file into file system.

Association with domain concept is

Very difficult	Difficult	Similar/the same as	Easy	Very easy
☐	☐	☐	☐	☐

when compared to saving product fragment to file system.

Q18. How would you describe the result using the CO_2SY tool?

Total disaster				*Very accurate*
1	2	3	4	5
☐	☐	☐	☐	☐

Q19. Imagine organizing your own information in a folder structure created by you. Would a tool such as the tested prototype (with conceptual model/ontology created by you) be a reasonable alternative for classifying your files?

Very unlikely	Unlikely	I do not know	Likely	Very likely
☐	☐	☐	☐	☐

Sometimes, when associating you will have a fragment where you will not find the exact names of concepts that are used in the model.
Q20. How easy was it to find a dependent product fragment?

Very difficult	Difficult	Moderate	Easy	Very easy
☐	☐	☐	☐	☐

Q21. How would you rate CO_2SY system, was it difficult to locate the concept?

Very difficult	Difficult	Moderate	Easy	Very easy
☐	☐	☐	☐	☐

Q22. If concept was not present in a model – was it difficult to decide which concept to use for association with a fragment?

Very difficult	Difficult	Moderate	Easy	Very easy
☐	☐	☐	☐	☐

Q23. Please specify what feature of the tested tools you liked most, and which tool:

Q24. Which of the tools/ techniques would you prefer to use in the future, and why:

☐ *Doors,*
☐ *CO_2SY,*
☐ *Traceability matrix*

Please provide the reason:

Q25. What you would like to add to the tested prototype (CO_2SY) and to the each of the other systems?

Doors

CO_2SY

Traceability matrix

Q26. In Case 2 you have been presented with a conceptual domain model, i.e. not only the list of concepts on the screen of CO_2SY. Please identify whether it was:

More difficult	A bit difficult	Moderate	Easier	Much easier
☐	☐	☐	☐	☐

to locate right concept than dealing only with list of concepts in case 1.

Q27. Other comments, remarks regarding the experiment itself, used tools and cases, procedure of the experiment, questionnaire (please specify)

Thank you
for the time used for the experiment and questionnaire!
Your help is very appreciated!

Appendix D
Experimental materials

This appendix presents the experimental material used for the experiment described in chapter 8. Next, the cases used in the experiment are presented, including both the descriptions and domain models. Then, the product fragments used in the experiment are listed and exemplified by inclusion of two typical product fragments for each product fragment type and the case.

D.1 Description of Case 1

This case describes a project implementing a system to support software development. A simplified model defining the main concepts in the given domain is presented for Case 1 in figure 1 below. A software development *project* has a development methodology defined as a *phase structure*, composed of different *phases* (e.g., requirements engineering, design, programming, etc.), every *development object* belongs to one of the phases and is stored in a *repository*. There are stakeholders (*users*) participating in a project. Users' *interaction* on development object creates an *event* of a certain *event type*. User can subscribe (has *subscription*) to certain event types in order to get *notification* about an event of his/her interest. *Creation*, *alteration* and *discussion* of development object are the main interactions. Development objects have dependency on each other; as well they are classified (have an *association*) with a *concept* from *domain model*.

In summary, the system has functionality as follows. User creates a new or alters an existing development object. After creation of new object a dependency to other objects is established. Further, development object is classified according to the domain model. User gets informed about creation or alteration of the object if he/she is subscribed to a particular type of event.

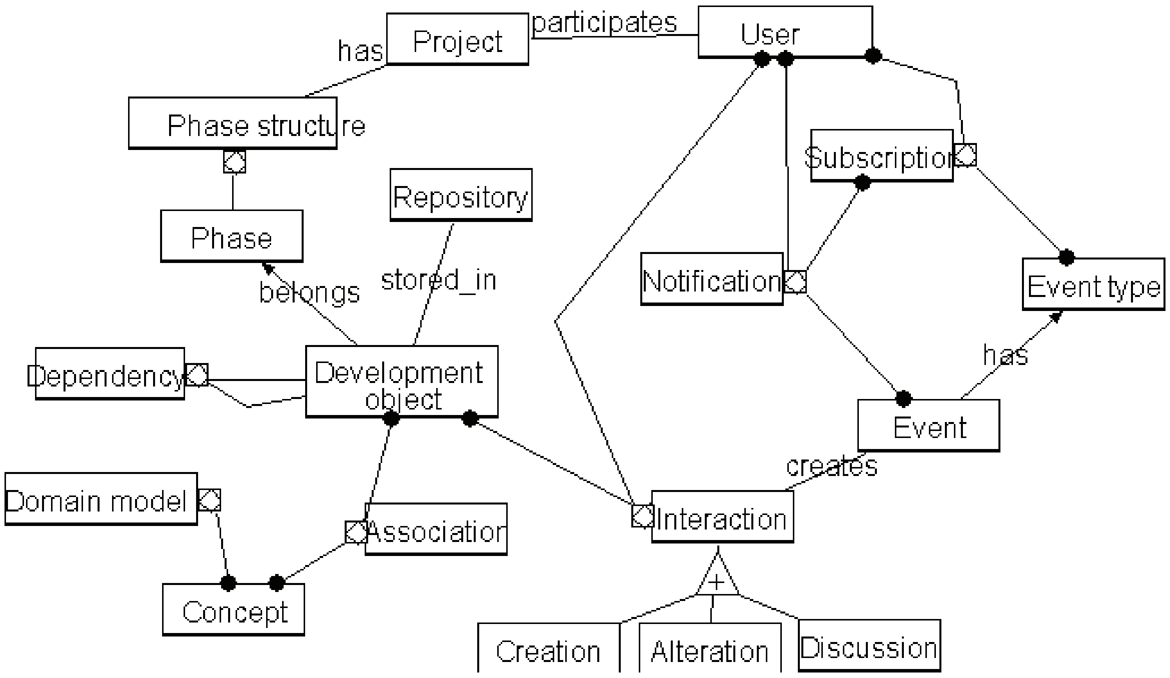


Figure D.1 Domain model for Case 1

D.1.1 Product fragments in case 1

There are four different types of development objects in this case. Namely, requirement statements, design fragments (UML sequence diagrams), code (C3) fragments and user manual in a form of screenshots with an explanation. Table D.1 lists all product fragments.

Table D.1 Product fragments in case 1

Product fragment		
Type	**Name**	**Description**
Design	SQD_Login	sequence diagram for logging to the system
	SQD_client_config	sequence diagram for client configuration
	SQD_message_handling	sequence diagram for message handling
	SQD_event_management	sequence diagram for event management
	SQD_Logout	sequence diagram for logging out from the system
	SQD_project_activity	sequence diagram defining general project activities
	SQD_fragment_creation	sequence diagram for fragment creation
	SQD_fragment_manipulation	sequence diagram for dev.object alteration
	SQD_model_alteration	sequence diagram defining domain model alteration
	SQD_event_notification	sequence diagram for event notification
	SQD_server_administration	sequence diagram defining server administraiton task
	SQD_server_operation	sequence diagram defining server operations

Product fragment		
Type	**Name**	**Description**
User manual	UM_login	screenshot of login window
	UM_MAin_window	screenshot of the main window
	UM_configure1	screenshot for system configuration (part 1)
	UM_configure2	screenshot illustrating system configuration (part2)
	UM_user_management	screenshot of the user management
	UM_fragment_manipulation	screenshot illustrating manipulation of development object
	UM_concept_manipulation	screenshot for concept definition
	UM_impact_analysis	screenshot of window with graphical information about impact
	UM_event_message	screenshot illustrating event message pop-up
	UM_message_handling	screenshot of the window for message handling
	UM_event_handling_history	screenshot illustrating event history handling
	UM_event_subscription	screenshot illustrating a subscription for event type
	UM_phase_definition	screenshot of the window for project phase definition
Code	ctlConcept	domain model concept presentation
	ctlConceptView	A view controller for containing and managing concepts and relations
	ctlFragment	Fragment representation in GUI
	ctlFragmentView	A view controller for containing and managing fragments, linkgroups and links.
	ctlImpact	Impact miniature sized representation of working board
	ctlLink	Link representation in GUI
	frmConcept	Manage concept information
	frmConfig	Server configuration
	frmEventPopup	Event popup notification window
	frmEvents	Manage events and subscription to events
	frmFragment	Manage fragment (dev.obj) information: view or revision
	frmImpactAnalysis	Management of impact analysis
	frmMain	main window and system tray application window. Starting point for any action.
	frmMessages	Manage messages
	frmProjects	Organizes the fragment (dev.obj) and domain model controls
	frmRelation	manage concept relation information
	frmUsers	User and group management. Alter users, groups and user's group membership.

Product fragment		
Type	**Name**	**Description**
Requirements	req1.1	General user should be able to browse, trace and find fragments effectively
	req1.1.1	General user should be able to ADD, VIEW, EDIT and DELETE fragments of which he/she has sufficient access to.
	req1.1.2	Interactor should be able to post a request for ADD to, VIEW,EDIT and DELETE from the reference-model.
	req1.1.3	Interactor should get satisfactory feedback from the system when changes occur (impact analysis management)
	req1.1.4	Interactor should be able to ADD, EDIT and DELETE mapping/links from fragments to the reference-model or other fragments.
	req1.1.6	Interactor should be able to work effectively in within the assigned projects, with a sufficient and reasonable amount of relevant information provided at all times.
	req1.2	General user should be able to communicate with other users when needed, through customized or standard message templates. Messages may be categorized as private or project related.
	req1.2.1	Interactor should be able to ADD, EDIT and DELETE projects
	req1.2.10	Manager should be able to retain an overview of the project progress at all times.
	req1.2.2	Manager should be able to ADD, EDIT and DELETE general users and their sub-types.
	req1.2.3	Manager should be able to define the phase structure and artefacts used in the project
	req1.2.4	Manager should be able to assign default CASE-tools for required project artifacts.
	req1.2.5	Manager should be able to set up locking scheme.
	req1.2.6	Manager should be able to set up version management scheme
	req1.2.7	Manager should be able to import, export or generate parts or the complete conceptual domain model
	req1.2.8	Manager should be able to ADD, EDIT and DELETE concepts
	req1.2.9	Manager should be able to view reference-model update requests from other users.
	req1.3	General user should be able request notification of chosen events
	req1.3.1	Observer should be able to browse project information and reports at different levels of detail, within level of access.

Product fragment		
Type	**Name**	**Description**
	req1.4	General user should be able to view log of events that have and are to occur
	req1.5	General user should be able to view history of messages sent and received, with context of message.
	req1.6	General user should be able to post and read messages on a PUBLIC message board.
	req1.7	General user should be able to acquire summarized project information within the access of the user. Project related messages among users are considered as a part of the change logging.
	req1.8	General user should be able to log on to personal tracker from any location
	req1.9	General user should be able launch CASE-tools required to view the fragments, if available to the user at the current location.
	req2.1	System should be able to browse project information and reports at different levels of detail, within level of access.
	req2.1.1	System should process and manage the traceability related information to and from users, at all times
	req2.1.2	System should facilitate the transmission of the necessary data from the repository to the clients requesting the information
	req2.1.3	System should authenticate users of connecting clients.
	req2.2	System should provide the users with relevant information at all times
	req2.2.1	System should relate relevant information to the user
	req2.2.2	System should present the relevant information in a form appropriate for the contents and urgency of the information, while being customizable for individual adaptation.
	req2.2.3	System should provide any information clearly and expressively through the user of expressive textual and graphical representation, i.e. the use of colour coding and images..
	req2.2.4	System should facilitate the transmission of relevant traceability information from the user to the other users.
	req2.2.5	System should provide the user with communication assistance when requested
	req2.2.6	System should provide the user with the possibility of requesting events when desired by the user
	req2.2.7	System should enable fragmentation of non-translatable fragment. Through a fragment-border scheme, where the borders on the object signifies different fragments.

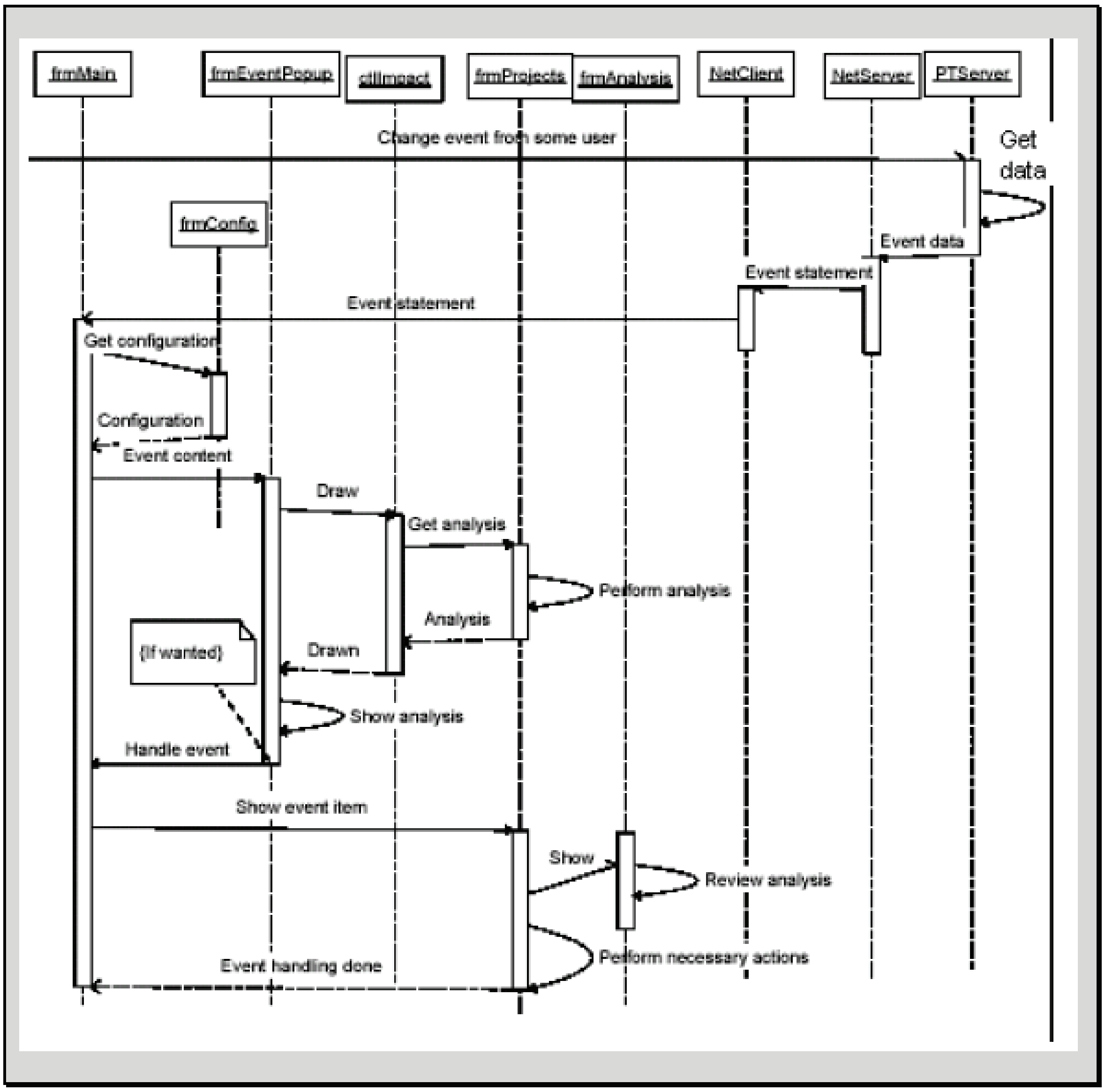
Product fragment		
Type	**Name**	**Description**
	req2.2.8	Colour coding should be used whenever appropriate to achieve maximum expressive power within limited screen space.
	req2.3.1	Repository should be able to STORE trace information and project information sent to the repository as fragments in XML.
	req2.3.2	Repository should be able to RETRIEVE trace information and project information requested from the repository
	req2.3.3	Repository should reliably and consistently store all information sent to it

Next, two typical examples of each product fragment type illustrates the above listed product fragments. For design fragments see Example D.1 and D2; user manual is illustrated in Example D.3 and D4; code fragments are illustrated in Example D.5 and D.6. The requirement fragments were base on the separate requirements statements, as listed in Table D.1.

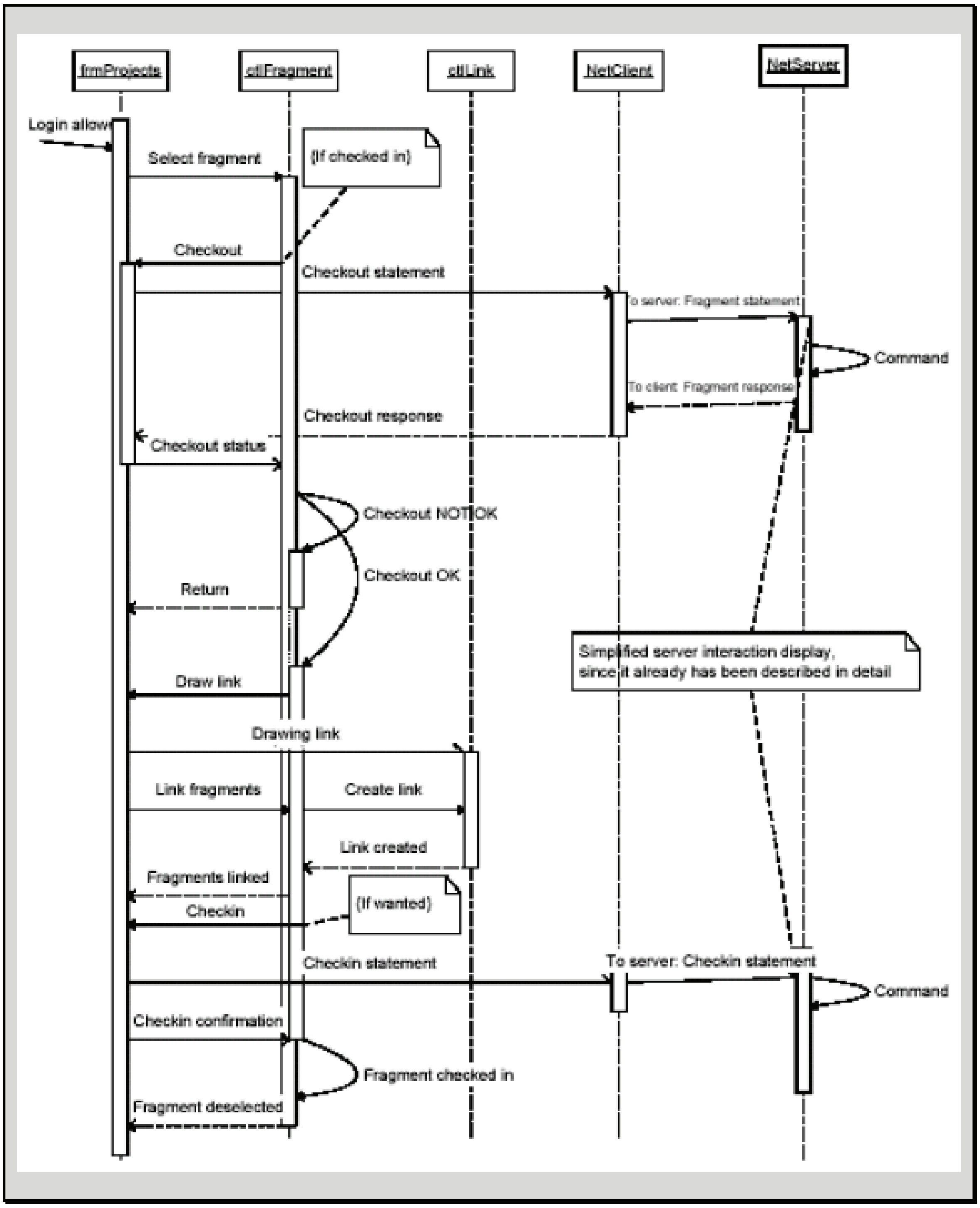
D.1.2 Illustration of product fragments in case 1

Design fragments

Example D.1 Case 1 fragment of design "SQD_Event_notification"

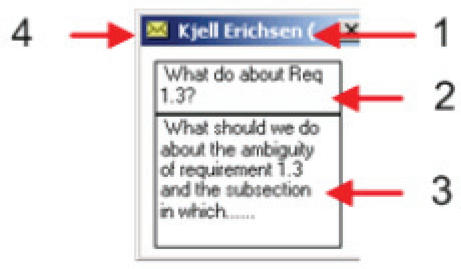


Example D.2 Case 1 fragment of design "SQD_fragment_manipulation"



User Manual

Example D.3 Case 1 fragment of user manual "UM_event_message"



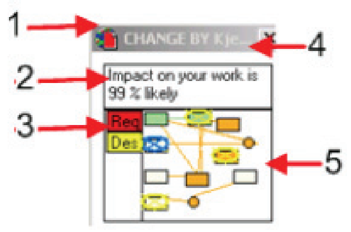
Kjell Erichsen (

What do about Req 1.3?

What should we do about the ambiguity of requirement 1.3 and the subsection in which......

Ref:	*Action:*	*Description*
1	Read	Originating user
2	Read	Message subject
3	Read	Beginning of message body
4	Read	Message event
4	2x Left-click	Bring up message handling for sending user

Example D.4 Case 1 fragment of user manual "UM_impact_analysis"



Impact on your work is 99 % likely

Ref:	*Action:*	*Description*
1	Read	Event severity: RED highest, YELLOW mid range, GREEN trivial
2	Read	Description of impact assessment and change type
3	Read	Affected phases coloured according to impact severity within phase
4	Read	Shows which user the change originates from
5	Read	Shows a miniview of affect project, with the changed fragment being marked in blue, while the affected fragments are marked in their respective severity colour; RED, YELLOW, GREEN
5	2x Left-click	Bring up project window where change has occurred

Code

Example D.5 Case 1 code (C#) fragment "ctlImpact"

```
using System;
using System.Collections;
using System.ComponentModel;
using System.Drawing;
using System.Data;
using System.Windows.Forms;

namespace PT
{
   /// <summary>
   /// Summary description for ctlImpact.
   /// </summary>
   public class ctlImpact : System.Windows.Forms.UserControl
   {
      /// <summary>
      /// Required designer variable.
      /// </summary>
      private System.ComponentModel.Container components = null;
      private ImpactAnalysis situation;

      public ctlImpact()
      {
         // This call is required by the Windows.Forms Form Designer.
         InitializeComponent();

         // TODO: Add any initialization after the InitForm call

      }

      public ctlImpact(ImpactAnalysis impact)
      {
         // This call is required by the Windows.Forms Form Designer.
         InitializeComponent();

         // TODO: Add any initialization after the InitForm call
         situation = impact;
      }

      /// <summary>
      /// Clean up any resources being used.
      /// </summary>
      protected override void Dispose( bool disposing )
      {
         if( disposing )
         {
            if(components != null)
            {
                  components.Dispose();
            }
         }
         base.Dispose( disposing );
      }
      #region Component Designer generated code
      /// <summary>
      /// Required method for Designer support - do not modify
      /// the contents of this method with the code editor.
      /// </summary>
      private void InitializeComponent()
      {
         //
         // ctlImpact
         //
         this.Name = "ctlImpact";
         this.Size = new System.Drawing.Size(80, 72);
         this.Load += new System.EventHandler(this.ctlImpact_Load);
         this.Paint += new
System.Windows.Forms.PaintEventHandler(this.ctlImpact_Paint);

      }
      #endregion

      public void setImpactAnalysis(ImpactAnalysis impact)
      {
         this.situation = impact;
         this.Invalidate();
      }

      private void ctlImpact_Paint(object sender,
System.Windows.Forms.PaintEventArgs e)
      {
         Graphics g = this.CreateGraphics();

         //Drawing outline of control
         g.DrawRectangle(new
Pen(Color.Black,2),0,0,this.Width,this.Height);

         //If there is any analysis to draw
         if(situation!=null)
         {

   situation.getProject().drawMiniView(this.Width,this.Height,g,situat
ion);
         }

      }

      private void ctlImpact_Load(object sender, System.EventArgs e)
      {

      }



   }
}
```

Example D.6 Case 1 code (C#) fragment "frmRelation"

```
using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;

namespace PT
{
   /// <summary>
   /// Summary description for frmRelation.
   /// </summary>
   public class frmRelation : PT.PTForm
   {
      private System.Windows.Forms.Label label1;
      private System.Windows.Forms.Label label2;
      private System.Windows.Forms.Label lblFrom;
      private System.Windows.Forms.Label lblTo;
      private System.Windows.Forms.Label label3;
      private System.Windows.Forms.Label label4;
      private System.Windows.Forms.TextBox txtRationale;
      private System.Windows.Forms.NumericUpDown numWeight;
      private System.Windows.Forms.Button btnSave;
      private System.Windows.Forms.Button btnDelete;
      private System.Windows.Forms.TextBox txtID;
      private ctlRelation relation;
      private frmProjects frmprojects;
      /// <summary>
      /// Required designer variable.
      /// </summary>
      private System.ComponentModel.Container components = null;

      public frmRelation(ctlRelation rel, frmProjects project)
      {
         //
         // Required for Windows Form Designer support
         //
         InitializeComponent();

         //
         // TODO: Add any constructor code after InitializeComponent
call
         //
         this.relation = rel;
         this.frmprojects = project;
      }

      /// <summary>
      /// Clean up any resources being used.
      /// </summary>
      protected override void Dispose( bool disposing )
      {
         if( disposing )
         {
            if(components != null)
            {
                components.Dispose();
            }
        }
        base.Dispose( disposing );
    }

    #region Windows Form Designer generated code
    /// <summary>
    /// Required method for Designer support - do not modify
    /// the contents of this method with the code editor.
    /// </summary>
    private void InitializeComponent()
    {
        System.Resources.ResourceManager resources = new
System.Resources.ResourceManager(typeof(frmRelation));
        this.label1 = new System.Windows.Forms.Label();
        this.label2 = new System.Windows.Forms.Label();
        this.lblFrom = new System.Windows.Forms.Label();
        this.lblTo = new System.Windows.Forms.Label();
        this.label3 = new System.Windows.Forms.Label();
        this.label4 = new System.Windows.Forms.Label();
        this.txtRationale = new System.Windows.Forms.TextBox();
        this.numWeight = new System.Windows.Forms.NumericUpDown();
        this.btnSave = new System.Windows.Forms.Button();
        this.btnDelete = new System.Windows.Forms.Button();
        this.txtID = new System.Windows.Forms.TextBox();

  ((System.ComponentModel.ISupportInitialize)(this.numWeight)).BeginI
nit();
        this.SuspendLayout();
        //
        // label1
        //
        this.label1.Location = new System.Drawing.Point(0, 8);
        this.label1.Name = "label1";
        this.label1.Size = new System.Drawing.Size(40, 16);
        this.label1.TabIndex = 0;
        this.label1.Text = "From:";
        //
        // label2
        //
        this.label2.Location = new System.Drawing.Point(0, 32);
        this.label2.Name = "label2";
        this.label2.Size = new System.Drawing.Size(32, 16);
        this.label2.TabIndex = 1;
        this.label2.Text = "To:";
        //
        // lblFrom
        //
        this.lblFrom.BorderStyle =
System.Windows.Forms.BorderStyle.Fixed3D;
        this.lblFrom.Location = new System.Drawing.Point(56, 8);
        this.lblFrom.Name = "lblFrom";
        this.lblFrom.Size = new System.Drawing.Size(216, 16);
        this.lblFrom.TabIndex = 2;
        //
        // lblTo
```

```
         //
         this.lblTo.BorderStyle =
System.Windows.Forms.BorderStyle.Fixed3D;
         this.lblTo.Location = new System.Drawing.Point(56, 32);
         this.lblTo.Name = "lblTo";
         this.lblTo.Size = new System.Drawing.Size(216, 16);
         this.lblTo.TabIndex = 3;
         //
         // label3
         //
         this.label3.Location = new System.Drawing.Point(0, 80);
         this.label3.Name = "label3";
         this.label3.Size = new System.Drawing.Size(56, 16);
         this.label3.TabIndex = 4;
         this.label3.Text = "Rationale:";
         //
         // label4
         //
         this.label4.Location = new System.Drawing.Point(0, 56);
         this.label4.Name = "label4";
         this.label4.Size = new System.Drawing.Size(48, 16);
         this.label4.TabIndex = 5;
         this.label4.Text = "Weight:";
         //
         // txtRationale
         //
         this.txtRationale.Location = new System.Drawing.Point(56,
80);
         this.txtRationale.Multiline = true;
         this.txtRationale.Name = "txtRationale";
         this.txtRationale.ReadOnly = true;
         this.txtRationale.Size = new System.Drawing.Size(208, 56);
         this.txtRationale.TabIndex = 6;
         this.txtRationale.Text = "";
         //
         // numWeight
         //
         this.numWeight.DecimalPlaces = 2;
         this.numWeight.Increment = new System.Decimal(new int[] {

            1,

            0,

            0,

            131072});
         this.numWeight.Location = new System.Drawing.Point(56, 56);
         this.numWeight.Maximum = new System.Decimal(new int[] {

           1,

           0,

           0,

           0});
         this.numWeight.Name = "numWeight";
         this.numWeight.ReadOnly = true;
         this.numWeight.Size = new System.Drawing.Size(48, 20);
         this.numWeight.TabIndex = 7;
         this.numWeight.Value = new System.Decimal(new int[] {

         1,

         0,

         0,

         0});
         //
         // btnSave
         //
         this.btnSave.Location = new System.Drawing.Point(72, 144);
         this.btnSave.Name = "btnSave";
         this.btnSave.Size = new System.Drawing.Size(64, 24);
         this.btnSave.TabIndex = 8;
         this.btnSave.Text = "Save";
         this.btnSave.Click += new
System.EventHandler(this.btnSave_Click);
         //
         // btnDelete
         //
         this.btnDelete.Enabled = false;
         this.btnDelete.Location = new System.Drawing.Point(152, 144);
         this.btnDelete.Name = "btnDelete";
         this.btnDelete.Size = new System.Drawing.Size(64, 24);
         this.btnDelete.TabIndex = 9;
         this.btnDelete.Text = "Delete";
         this.btnDelete.Click += new
System.EventHandler(this.btnDelete_Click);
         //
         // txtID
         //
         this.txtID.Location = new System.Drawing.Point(200, 56);
         this.txtID.Name = "txtID";
         this.txtID.Size = new System.Drawing.Size(64, 20);
         this.txtID.TabIndex = 10;
         this.txtID.Text = "";
         this.txtID.Visible = false;
         //
         // frmRelation
         //
         this.AutoScaleBaseSize = new System.Drawing.Size(5, 13);
         this.ClientSize = new System.Drawing.Size(280, 173);
         this.Controls.AddRange(new System.Windows.Forms.Control[] {

              this.txtID,

              this.btnDelete,

              this.btnSave,

              this.numWeight,
              this.txtRationale,

              this.label4,

              this.label3,

              this.lblTo,

              this.lblFrom,

              this.label2,

              this.label1});
         this.Icon =
((System.Drawing.Icon)(resources.GetObject("$this.Icon")));
         this.MaximizeBox = false;
         this.MinimizeBox = false;
         this.Name = "frmRelation";
         this.SizeGripStyle = System.Windows.Forms.SizeGripStyle.Hide;
         this.Text = "Relation";
         this.Load += new System.EventHandler(this.frmRelation_Load);

   ((System.ComponentModel.ISupportInitialize)(this.numWeight)).EndIni
t();
         this.ResumeLayout(false);

      }
      #endregion

      private void stop()
      {
         frmprojects.Activate(); //Returning focus
         this.Dispose();

      }

      private void frmRelation_Load(object sender, System.EventArgs e)
      {

         XmlHolder holder  = this.relation.getHolder();

         //Setting holding controllers
         this.lblFrom.Text = relation.getControlFrom().getID();
         this.lblTo.Text = relation.getControlTo().getID();

         //If this is a NEW relation, enable some more boxes
         //AND if the user has the metamodel checked out
         if(holder==null )
         {

   if(FrmMain.medi.getPTUser().getID().CompareTo(this.frmprojects.getP
roject().getTag("checkoutbyid"))==0)
            {   //Enabling save button
               this.btnSave.Visible = true;
               this.btnSave.Enabled = true;
               this.btnDelete.Enabled = false;
               //Enable writing of name
               this.numWeight.ReadOnly = false;
               this.txtRationale.ReadOnly = false;
            }
         }
         else //if this is an already existing concept, show
information
         {

            //Disable save button

            //TODO: update link somehow, selection boxes for FROM and
TO?
            this.btnSave.Visible = false;
            this.btnSave.Enabled = true;
            this.btnSave.Text = "Save changes";

            //If user has metamodel checked out, enable delete

   if(FrmMain.medi.getPTUser().getID().CompareTo(this.frmprojects.getP
roject().getTag("checkoutbyid"))==0)
               this.btnDelete.Enabled = true;

            //Disable writing of name
            this.numWeight.ReadOnly = true;
            this.txtRationale.ReadOnly = true;

            //Setting relation information
            this.txtID.Text = holder.getTag("id");
            this.txtRationale.Text = holder.getTag("rationale");
            this.numWeight.Value =
Convert.ToDecimal(holder.getTag("weight"));
         }


      }


      //Saving or updating the fragment
      private void btnSave_Click(object sender, System.EventArgs e)
      {

         Cursor.Current = Cursors.WaitCursor; //Setting wait cursor

         //Save relation info
         string cmdword_begin  = "";
         string cmdword_end  = "";
         string cmdmsg = "";

         //If the button says SAVE it's a NEW relation
         if(this.btnSave.Text.CompareTo("Save")==0)
         {
            cmdword_begin = "<NEW respond=\"true\">";

            cmdword_end = "</NEW>";
            cmdmsg = "added";
         }
         else  //Otherwise, it's an UPDATE
         {
            cmdword_begin = "<UPDATE respond=\"true\">";
            cmdword_end = "</UPDATE>";
            cmdmsg = "updated";
         }

         string msg = "<PTSERVER><DATA>"+ cmdword_begin +
"<RELATION>";
         msg += "<ID>" + this.txtID.Text + "</ID><WEIGHT>" +
this.numWeight.Value.ToString() + "</WEIGHT><RATIONALE>"+
this.txtRationale.Text + "</RATIONALE>";
         msg += "<FROMCONCEPTID>"+this.lblFrom.Text
+"</FROMCONCEPTID><TOCONCEPTID>"+this.lblTo.Text
+"</TOCONCEPTID><CREATEDBYID>" + PT.PTForm.medi.getPTUser().getID()+ "
</CREATEDBYID>";
         msg += "</RELATION>"+ cmdword_end + "</DATA></PTSERVER>";
         //Send update wait for ID and set it in holder
         XmlHolder[] answer = this.netGetArrayOfItems(msg,"relation");
         //If we got a valid response
         if(answer.Length>0)
         {

            //Remove old

   frmprojects.getConceptView().removeRelation(relation.getID());

            //Setting new relation information
            relation.setHolder(answer[0]);

            //Registering new relation with view
            frmprojects.getConceptView().registerRelation(relation);

            //Display ok message
            PT.PTForm.medi.displayOK("Relation was
"+cmdmsg+".",cmdmsg);

            this.stop(); //Closing
         }
         else //Display error message
            PT.PTForm.medi.displayError("Relation was NOT
"+cmdmsg+".");


         Cursor.Current = Cursors.Default; //Setting default
      }



      //When the user wants to delete relation
      private void btnDelete_Click(object sender, System.EventArgs e)
      {
         //Set relation as obsolete
         string msg = "<PTSERVER><DATA><UPDATE
respond=\"true\"><relation>";
```

```
         msg += "<ID>" + this.txtID.Text +
"</ID><OBSOLETE>1</OBSOLETE>";
         msg += "</relation></UPDATE></DATA></PTSERVER>";

         //Send update wait for ID and set it in holder
         XmlHolder[] answer = this.netGetArrayOfItems(msg,"relation");
         //If we got a valid response
         if(answer.Length>0)
         {
            //Setting new relation information
            relation.setHolder(answer[0]);

   frmprojects.getConceptView().removeRelation(relation.getID());
                relation.Dispose();

            //Display ok message
            PT.PTForm.medi.displayOK("Relation was
obsoleted.","Obsolete");

            frmprojects.getConceptView().Invalidate(); //Update
display
            this.stop(); //Closing

         }
         else //Display error message
            PT.PTForm.medi.displayError("Relation was NOT
obsoleted.");

      }

   }
}
```

D.2 Description of Case 2

This case describes a project developing a system at NTNU to be used to support exercise delivery, review and evaluation during for a particular courses. A simplified model for Case 2 is depicted in figure 1 for the purpose of defining the scope.

User is involved (takes part) in the *course*. There are three types of users, namely, *student*, *lecturer*, and *sensor*. Every user of the system has a *user profile*. There are two types of *delivery* in the course, i.e., *exercise* and *review*. Each delivery has a *deadline*.

Students are organized in *student groups* to deliver an exercise. Solution to an exercise usually consists of a *description* and a *model*. After delivering the exercise students are arranged into *review groups*, in order to peer-review delivered exercises.

Lecturer and sensor perform *evaluation* of both exercise and review of exercise, by assigning a *grade* and providing some feedback (*comment*).

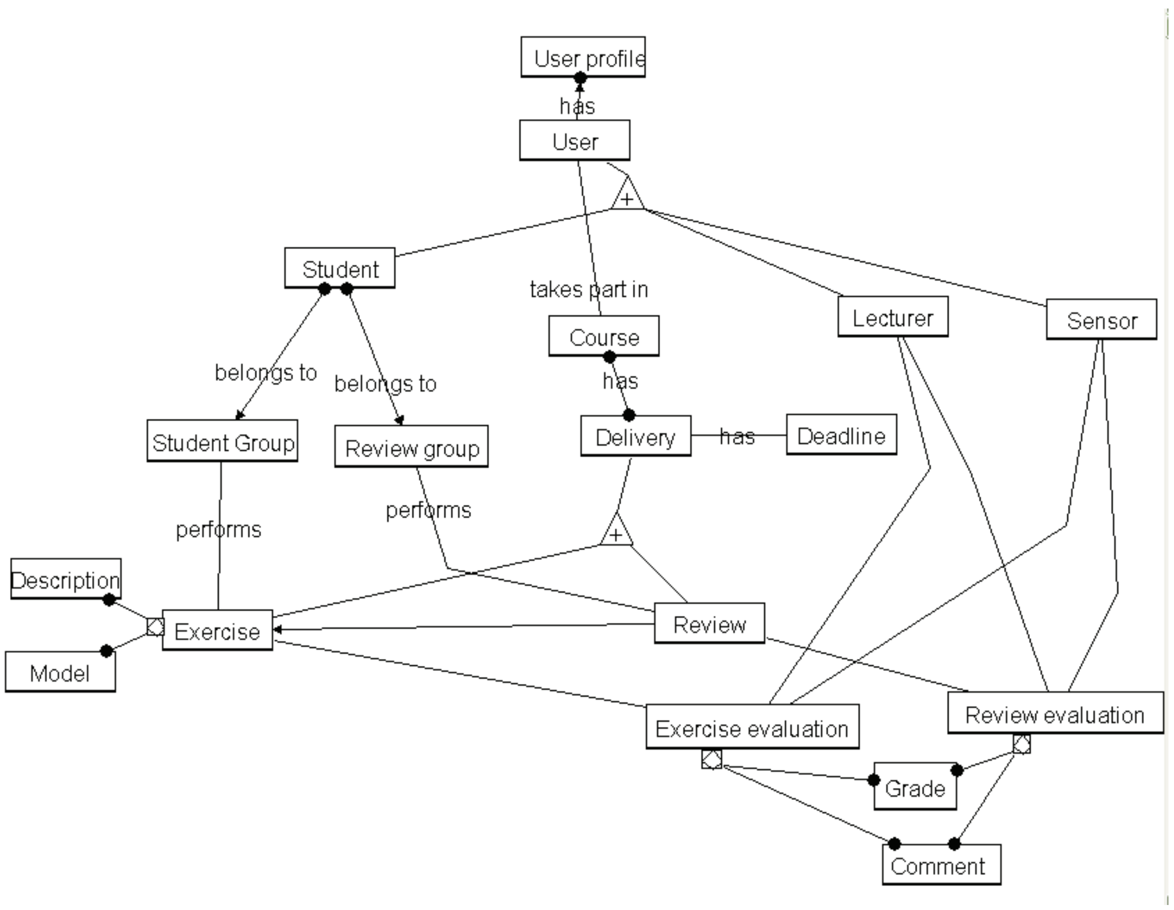


Figure D.2 Domain model for Case 2

D.2.1 Product fragments in case 2

There are two different types of development objects in this case. Namely, requirement statements, and code (php) fragments. Table D.2 lists the product fragments used in case 2.

Table D.2 Product fragments in case 2

Product fragment		
Type	**Name**	**Description**
Code	deliver	control of delivery form
	deliveries	overview of deliveries
	doEvaluate	control of evaluation form
	evaluate	control of reviewing form
	feedback	control of feedback form
	index	start form (login)
	updateReview	form for changing the review
	updateUserinfo	form for user profile update
	upload	exercise upload form and control

Product fragment		
Type	**Name**	**Description**
	userinfo	form for browinsg user profile
	viewDeliveryComments	form for browsing delivery evaluation (from student view)
	viewEvaluation	form for browsing evaluation of delivery
	viewFeedback	form for viewing received feedback (from lecturer view)
	viewReviewComments	form for browsing received review comments (from student view)
	editDeadline	form for setting and changing the deadlines
	viewGroups	set and browse students arranged into groups
Requirements	req3	Student should be able to log in to the system
	req5	Student should be able upload delivery (exercise)
	req6	Lecturer should be able to see the status of the deliveries from the assigned students.
	req8	Student should be able see the comments for solution
	req9	Lecturer should form a reviewer groups for delivery
	req11	Reviewer should be able evaluate the deliveries

Next, two typical code fragment in case 2 are illustrated.

Code

Example D.7 Case 2 code(php) fragment "deliver"

```
<?php $heading="Levere øving";
      $page = "deliver";
      require("header.php"); ?>
<!-- meny venstre -->
<?php require("../datoer.php");
echo "<br><br><table align=center><tr><td align=left>";
//Get urls to help documents for the model types
$models = mysql_query("select Name, DescriptionURL from
ModelType",$DB);
while ($row = mysql_fetch_array($models))
      echo "<a href=\"$row[DescriptionURL]\" style=\"font-
size:11px\">Om $row[Name]-diagrammer</a><br><br>";
?></td></tr></table>
</td>

<!-- hovedvindu -->
<td width="720" class="hoved" valign="top">
<div class="hovedvindu">

<?php
$op = $HTTP_POST_VARS[op];
if ($op == "redeliver") {
   //Get name of exercise to be delivered
   $result = mysql_query("select E.Name, GD.Deadline from Exercise as
E, GroupDeadline as GD where E.ExerciseID =
$HTTP_POST_VARS[redeliverID] and E.ExerciseID = GD.ExerciseID",$DB);
   if ($info = mysql_fetch_array($result))
      $exercisename = $info[Name];
   else {

   header("Location:$rot/student/deliveries.php?message=Finner+ikke+øv
ing!");
      exit;
   }?>
   <h1>Levér <? echo $exercisename ?> på nytt</h1><br>
   <span style="color:red">NB! Innleveringsfristen er endelig, og
muligheten for å levere stenger <br> automatisk kl 23:59 den <?php
echo fromMysql($info[Deadline]); ?>.</span><br><br>
      Det er kun mulig å levere én fil per oppgave. Vennligst levér
rapporten <br>i <i>pdf</i> eller <i>doc</i> format. Filen kan ikke
være større enn 10 MB.<br><br>
   <form action="upload.php" enctype="multipart/form-data"
method="post">
      <input type="hidden" name="MAX_FILE_SIZE" value="12000000">
      <input type="hidden" name="oving" value="<? echo
$HTTP_POST_VARS[redeliverID]; ?>">
      <input type="hidden" name="op" value="redeliver">
      Fil: &nbsp;<input name="userfile" type="file"><br><br>
      <input type="submit" name="submit" value="Lever fil">
   </form>
      <?php

} else {
   //Normal delivery
   $today = date("Y-m-d");
   //Get active exercise
   $findcurrent = mysql_query("select E.ExerciseID, E.Name,
GD.Deadline from GroupDeadline as GD, Exercise as E where
GD.StudGroupID = $HTTP_SESSION_VARS[StudGroupID] and E.Deadline >=
'$today' and E.ExerciseID = GD.ExerciseID order by GD.ExerciseID
asc",$DB);
   if ($denne = mysql_fetch_array($findcurrent)) {
      //Still exercises to deliver
      $delivered = mysql_query("select * from Delivery where
StudGroupID = $HTTP_SESSION_VARS[StudGroupID] and ExerciseID =
$denne[ExerciseID]",$DB);
      if (mysql_num_rows($delivered) != 0) {
         //Has delivered this exercise
         echo "<h1>Levere øving</h1><br>Du har levert $denne[Name].
<form action=deliver.php method=post style=\"margin:0\"><input
type=hidden name=op value=redeliver><input type=hidden
name=redeliverID value=$denne[ExerciseID]><br>Du kan levere den på
nytt her: <input type=submit value=\"Lever ny fil\"></form><br>";
         $nextexerciseid = $denne[ExerciseID] + 1;
         $findnext = mysql_query("select ExerciseID, Name from
Exercise as E where ExerciseID = $nextexerciseid",$DB);
         if ($neste = mysql_fetch_array($findnext))
            //There are more exercises to be delivered
            echo "Levering av $neste[Name] åpner etter
".fromMysql($denne[Deadline]).".";
```

```
         else
            //Last exercise delivered
            echo "Du har levert alle øvingene.";
      } else {
         //Shall deliver this exercise next
         ?>
         <h1>Levér <? echo $denne[Name]; ?></h1><br>
         <span style="color:red">NB! Innleveringsfristen er endelig,
og muligheten for å levere stenger <br> automatisk kl 23:59 den <?php
echo fromMysql($denne[Deadline]); ?>.</span><br><br>
         Det er kun mulig å levere én fil per oppgave. Vennligst levér
rapporten <br>i <i>pdf</i> eller <i>doc</i> format. Filen kan ikke
være større enn 10 MB.<br><br>

         <form action="upload.php" enctype="multipart/form-data"
method="post">
            <input type="hidden" name="MAX_FILE_SIZE"
value="12000000">
            <input type="hidden" name="oving" value="<?php echo
$denne[ExerciseID]?>">
            <input type="hidden" name="op" value="deliver">
            Fil: &nbsp;<input name="userfile" type="file"><br><br>
            <input type="submit" value="Lever fil">
         </form><?php
      }
   } else
      //Last deadline is passed
      echo "<h1>Levere øving</h1><br>Levering av siste øving er
avsluttet";
}
?><br><br>
<a href="deliveries.php">Se tidligere innleveringer</a>
</div>
</td>
<!-- hovedvindu slutt -->
<?php require("../footer.php"); ?>
```

Example D.8 Case 2 code (php) fragment "updateUserInfo"

```
<?php $rolle = "student";
include ("../dbconnect.php");

if ($op == "changeemail") {
   //User wants to update his email address. Test if he has actually
given a new address
   if ($newemail == '')
      $message = "Ingen epostadresse angitt!";
   else {
      //Update user info.
      mysql_query("update StudOnGroupset Email = '$newemail' where
StudGroupID = $HTTP_SESSION_VARS[StudGroupID]",$DB);
      $Email = $newemail;
      session_register("Email");
      $message = "Ny epostadresse lagret!";
   }
   session_register(message);
   header("Location:$rot/student/userinfo.php");
   exit;
} elseif ($op == "changepw") {
   //User wants to change password.
   if ($newpw == '')
      $message = "Du må angi et nytt passord!";
   else {
      //Get old password
      $result = mysql_query("select Password from StudGroup where
StudGroupID = $HTTP_SESSION_VARS[StudGroupID]",$DB);
      $row = mysql_fetch_row($result);
      //Test if given old password is correct before updating
password.
      if ($row[0] == $oldpw) {
         $result = mysql_query("update StudOnGroupset Password =
'$newpw' where StudGroupID = $HTTP_SESSION_VARS[StudGroupID]",$DB);
         $message = "Nytt passord lagret!";
      } else
         $message = "Feil med angitt passord!";
   }
   session_register(message);
   header("Location:$rot/student/userinfo.php");
   exit;
}?>
```

Appendix E
Data Collected

This appendix presents the raw material (i.e. responses to the questionnaire presented Appendix C) of the experiment described in chapter 8.

Background Data

Participants Summary

	Cases					
	Valid		Missing		Total	
	N	Percent	N	Percent	N	Percent
For all questions*	6	100,0%	0	,0%	6	100,0%

* - except Q20 and Q26, only five participants answered.

Q1. Gender

	Observed N
Male	3
Female	3
Total	**6**

Q2. Age

		Responses		Percent of Cases
		N	Percent	
$age(a)	21-30 years	2	33,3%	33,3%
	31-40 years	4	66,7%	66,7%
Total		**6**	**100,0%**	**100,0%**

Q3. Have you ever been involved in an industrial software development project? Check all that apply.

	Responses		Percent of Cases
	N	Percent	
I've been observing a software development project	3	27,3%	50,0%
I've participated in small software development project	4	36,4%	66,7%
I've participated in big software development project	3	27,3%	50,0%
I've participated in big geographically distributed software development project	1	9,1%	16,7%
Total	**11**	**100,0%**	**183,3%**

Q4. Have you ever been involved in usability evaluations? Check all that apply.

	Responses		Percent of Cases
	N	Percent	
No, I've never done anything like that	2	18,2%	33,3%
I've been observing a software development project	3	27,3%	50,0%
I've participated in small software development project	1	9,1%	16,7%
I've participated in big software development project	2	18,2%	33,3%
I've participated in big geographically distributed software development project	3	27,3%	50,0%
Total	**11**	**100,0%**	**183,3%**

Q5. How long you are (working) in the field of computer and information science

	Responses		Percent of Cases
	N	Percent	
6-10 years	1	16,7%	16,7%
11-25 years	5	83,3%	83,3%
Total	**6**	**100,0%**	**100,0%**

Evaluation Tasks and Questions

Q6. Please select random 3 (from marked ones) development object of any type in CO$_2$SY, select 3 top ranked values (lowest value) and check how many of them are correctly estimated as being related/dependent. Please do that for each of the cases.

How many of these would you rate as correct?

Case 1 / **Case 2**

Subject ID	Case 1 Wrong	Case 1 Partial	Case 1 Correct	Case 1 Total	Case 2 Wrong	Case 2 Partial	Case 2 Correct	Case 2 Total
1	0	3	6	**9**	2	3	4	**9**
2	4	3	4	**11**	3	2	3	**8**
3	5	8	15	**28**	1	3	7	**11**
4	2	0	15	**17**	5	3	5	**13**
5	6	1	7	**14**	1	2	8	**11**
6	6	1	5	**12**	9	1	5	**15**
Total	**23**	**16**	**52**	**91**	**21**	**14**	**32**	**67**

Q7. Browse the results of association and linking (all tools), compare them. Please identify, whether CO_2SY help to discover any new correct relatedness/ dependency relationship between development objects.

	Yes, but all were wrong	None	Neither, nor…	Some	Many
Observed N	-	-	-	6	-

Approach Evolution

Q8. What is your experience with:

Responses about Doors. Frequencies

Doors	Responses		Percent of Cases
	N	Percent	
Never used	6	100%	100%
Seen but not used	0	0%	0%
Used a few times	0	0%	0%
Used extensively	0	0%	0%
Total	**6**	**100,0%**	**100,0%**

Responses about CO_2SY.

CO_2SY	Responses		Percent of Cases
	N	Percent	
Never used	4	66,7%	66,7%
Seen but not used	2	33,3%	33,3%
Used a few times	0	0%	0%
Used extensively	0	0%	0%
Total	**6**	**100,0%**	**100,0%**

Responses about traceability matrix.

Traceability matrix	Responses		Percent of Cases
	N	Percent	
Never used	4	66,7%	66,7%
Seen but not used	1	16,7%	16,7%
Used a few times	1	16,7%	16,7%
Used extensively	0	0%	0%
Total	**6**	**100,0%**	**100,0%**

Q9. Please provide your opinion about the quality of case description:

	very bad	bad	fair	good	very good
Case 1 description	0	1	4	1	0
Case 1 domain model	0	0	4	2	0
Case 1 product fragments	0	1	3	1	1
Case 2 description	0	0	2	4	0
Case 2 domain model	0	0	1	5	0
Case 2 product fragments	0	2	2	2	0

Q10. Please rank used tools / techniques based on perceived easiness and efficiency using it for this particular settings:

Subject ID	**1**	**2**	**3**	**4**	**5**	**6**	**Average**
Prototype	1	1	1	2	1	1	*1,17*
Doors	3	2	2	1	2	3	*2,17*
Traceability matrix	2	3	3	3	3	2	*2,67*

Q11. Imagine a geographically distributed software development project bigger 10 times or more than the specified in the case 1. Please rank used tools / techniques based on perceived easiness and efficiency, but in the settings of big software development:

Subject ID	**1**	**2**	**3**	**4**	**5**	**6**	**Average**
Prototype	1	1	1	1	1	1	*1,00*
Doors	3	2	2	2	2	2	*2,17*
Traceability matrix	2	3	3	3	3	3	*2,83*

Q12. When associating design and code fragments, did you experienced difficulty while choosing concept from domain model? If so, do you think it was because of (choose that apply):

	Responses		Percent of Cases
	N	Percent	
No, I have not experienced any difficulty	1	11,1%	16,7%
Lack of design/code oriented concepts	1	11,1%	16,7%
Lack of domain knowledge	3	33,3%	50,0%

	Responses		Percent of Cases
	N	Percent	
Other	4	44,4%	66,7%
Total	**9**	**100,0%**	**150,1%**

Results whether lack of knowledge was for case 1, case 2 or both:

	Responses		Percent of Cases
	N	Percent	
Case 1	3	75,0%	100,0%
Case 2	1	25,0%	33,3%
Total	**4**	**100,0%**	**133,3%**

Comments provided by the subjects who selected *Other* reason for having difficulty while choosing concept from domain model:

Subject id	Comment
1	Poorly commented code and design.
2	Insufficient study of fragments.
4	Mixed up by concepts and functions/ workflow.
5	The concepts should have been more defined.

Q13. In general how would you rate the user interface of:

Doors	General ease of use	Visual Design / Layout	Ease of Learning
	Observed N	Observed N	Observed N
Needs major improvements	2	1	0
Needs minor improvement	1	3	2
Fair	2	1	3
Good	1	0	1
Works well	0	1	0
Total	**6**	**6**	**6**

CO_2SY	General ease of use	Visual Design / Layout	Ease of Learning
	Observed N	Observed N	Observed N
Needs major improvements	0	0	0
Needs minor improvement	0	5	0
Fair	5	1	1
Good	1	0	5
Works well	0	0	0
Total	**6**	**6**	**6**

Q14. Are you satisfied with a linking functionality (for the purpose of identifying related/dependent items) of the tested tools/ techniques? Please use a scale from 1 to 5.

	Q14_Doors Observed N	Q14_CO$_2$SY Observed N	Q14_trace matrix Observed N
Not at all	2	0	0
.	1	0	3
.	1	1	1
.	2	3	0
Very satisfied	0	2	2
Total	**6**	**6**	**6**

Q15. Please evaluate the usability of the tools (usability considers how easy it is to learn and use a tool):

	Q15_Doors Observed N	Q15_ CO$_2$SY Observed N	Q15_ trace matrix Observed N
Very difficult	0	0	1
Difficult	2	0	0
Moderate	3	3	1
Easy	1	2	1
Very easy	0	1	3
Total	**6**	**6**	**6**

Q16. Please evaluate easiness of dependency establishment using the tools (consider how labour intensive was the establishment of links/associations):

	Q16_Doors Observed N	Q16_ CO$_2$SY Observed N	Q16_trace matrix Observed N
Very difficult	2	0	0
Difficult	1	0	2
Moderate	2	1	0
Easy	1	5	2
Very easy	0	0	2
Total	**6**	**6**	**6**

Q17. Imagine a project relying on a well structured folder hierarchy in file system. The option would be to save file in corresponding folder, e.g., Project_name/User/Phase/... Please compare association with domain concept to saving file into file system.

Association with domain concept is

	Responses		Percent of Cases
	N	Percent	
Very difficult	0	0%	0%
Difficult	0	0%	0%
Similar/ the same as	1	16,7%	16,7%

	Responses		Percent of Cases
	N	Percent	
Easy	5	83,3%	83,3%
Very easy	0	0%	0%
Total	**6**	**100,0%**	**100,0%**

when compared to saving product fragment to file system.

Q18. How would you describe the result using the CO_2SY tool?

	Responses		Percent of Cases
	N	Percent	
1. Total disaster	0	0%	0%
2.	0	0%	0%
3.	2	33,3%	33,3%
4.	4	66,7%	66,7%
5. Very accurate	0	0%	0%
Total	**6**	**100,0%**	**100,0%**

Q19. Imagine organizing your own information in a folder structure created by you. Would a tool such as the tested prototype (with conceptual model/ontology created by you) be a reasonable alternative for classifying your files?

	Responses		Percent of Cases
	N	Percent	
Very unlikely	0	0%	0%
Unlikely	0	0%	0%
I do not know	1	16,7%	16,7%
Likely	3	50,0%	50,0%
Very likely	2	33,3%	33,3%
Total	**6**	**100,0%**	**100,0%**

Sometimes, when associating you will have a fragment where you will not find the exact names of concepts that are used in the model.
Q20. How easy was it to find a dependent product fragment?
Q21. How would you rate CO_2SY system, was it difficult to locate the concept?
Q22. If concept was not present in a model – was it difficult to decide which concept to use for association with a fragment?

	Q20		Q21		Q22	
	N	Percent	N	Percent	N	Percent
Very difficult	0	0%	0	0%	0	0%
Difficult	2	40,0%	0	0%	2	33,3%

	Q20		Q21		Q22	
	N	Percent	N	Percent	N	Percent
Moderate	0	0%	3	50,0%	1	16,7%
Easy	3	60,0%	3	50,0%	3	50,0%
Very easy	0	0%	0	0%	0	0%
Total	**5**	**100,0%**	**6**	**100,0%**	**6**	**100,0%**

Q23. Please specify what feature of the tested tools you liked most, and which tool:

Subject id	Comment
1	*CO_2SY*: To have the linked items and link specification dialogue box in the same window, would have been nice to have had graphic representation of the domain model, though. With a possibilty to display NL definition when needed.
2	*CO_2SY*: Discovery of mismatch between associations and direct linking.
3	*Doors*: the visualization of the link results. *CO_2SY*: the automatic estimation of the link results. *Traceability matrix*: easy to learn.
4	*Doors*: Trace interface. *CO_2SY*: The function of mapping code fragments, etc to concept model is appreciated.
5	*CO_2SY*: seeing dependencies found by the system. But could have been graphical.
6	*CO_2SY*: Relating artifacts to domain model concepts using the tool -> evaluating the correctness of the decisions.

Q24. Which of the tools/ techniques would you prefer to use in the future, and why:

Subject id	Comment
1	*CO_2SY*: Traceability matrix is easy to use, but do not allow to grade the strength of the association, nor does it do the hard work (finding the specific domain objects) for me. relating the fragments to model concepts was more intuitive and fuzzy enough. The prototype would even have corrected my mistake! Thus, based on this limited experience, I find it trustworthy, even though I'm not in control of the detailed links. Apparently saves a lot of time!
2	*CO_2SY* and *Traceability matrix.*
3	*CO_2SY*: I need only link the fragments to the domain concepts in which the amount of concepts is smaller than the number of all fragments. It seems I used less efforts and the dependencies between fragments can be estimated automatically.
4	*Doors* and *CO_2SY.* Doors is more suitable for moderate size and structured projects. CO_2SY will be great for large scale, distributed development which may contain plentiful of fragments.
5	*CO_2SY*: Found it easier to use and more intelligent.
6	*CO_2SY*: because novel approach and ease of use.

Q25. What you would like to add to the tested prototype (CO_2SY) and to the each of the other systems?

Subject id	Comment
1	*Doors*: Much less fragmented user interface and many less steps required when specifying a link! *CO_2SY*: Visualization of the model and also of the closesness analysis. Sometimes I would felt more secure to be able to mark “absolutely not linked to” for some concepts. *Traceability matrix*: Grading of confidentiality in links.
2	*CO_2SY*: Better GUI (more intuitive with undo).
3	*Doors*: quick navigation function; I don’t like layered windows when I link different fragments in different windows. *CO_2SY*: Need more comprehensive link result specification. Need to provide different views and filters to see the results. *Traceability matrix*: It will be too big if a case is big. Need a function of filter.
4	*Doors*: Concept mapping support. *CO_2SY*: functions in Doors (code to code, code to requirements, etc). *Traceability matrix*: get rid of it if having the other two.
5	See Q23.
6	I can’t comment about functionality at this stage – but all of them need to convince me why you would need to use this idea. What about scalability?

Q26. In Case 2 you have been presented with a conceptual domain model, i.e. not only the list of concepts on the screen of CO_2SY. Please identify whether it was:

	Responses		Percent of Cases
	N	Percent	
More difficult	0	0%	0%
A bit difficult	0	0%	0%
Moderate	0	0%	0%
Easy	4	80,0%	66,7%
Much easier	1	20,0%	16,7%
Total	**5**	**100,0%**	**83,3%**

to locate right concept than dealing only with list of concepts in case 1.

Q27. Other comments, remarks regarding the experiment itself, used tools and cases, procedure of the experiment, questionnaire (please specify)

Subject id	Comment
1	In Q9, quality of models, my rating would be higher, if the used modeling notation had been more intuitive. In Q26 I mean I never looked at the list expect when assigning the association.
2	Too generic requirements in case 1. Unrealistic estimation of duration.
3	The experiment itself and used tool are sound and enough. The procedure is also fine just a little long. The case 1 is not easy to understand for some testers. I think case 2 is more comprehensive and understandable for normal testers. For questionnaire. Some questions are not specified explicitly, e.g. Q17. Some

Subject id	Comment
	questions can not be answered for sure because it relates to the all functions or performance of the tool (e.g. Q15).
4	-
5	-
6	Maybe a little more explanation for why need to use such a tool would be good aspect of training.

Appendix F
Collection of Papers

This appendix includes the papers publish as a part of this thesis and described in chapter 1.6 of the thesis. They are as follows.

F.1 Traceability in Collaborative Systems Development from Lifecycle Perspective - a position paper

Strasunskas D. Traceability in Collaborative Systems Development from Lifecycle Perspective - a position paper. In *Proc. of the 1st Intl. Workshop on Traceability* (TEFSE), co-located with ASE 2002, Edinburgh, Scotland, UK, September 2002, pages 54-60.

Traceability in Collaborative Systems Development from Lifecycle Perspective – A Position Paper

Darijus Strašunskas
Dept. of Computer and Information Science, Norwegian Univ. of Science and Technology
Sem Sælands vei 7-9, NO-7491 Trondheim, Norway
dstrasun@idi.ntnu.no

Abstract

The aim of this position paper is to discuss the features of state-of-the-art and outstanding issues of the traceability between product fragments in collaborative system development.

A lot of research has been done in the pre-traceability area. Recently, researchers' attitudes towards inter-relation of requirements and architecture elements have increased. Several approaches to tackle this problem have been proposed. Nevertheless, to the author's knowledge, the solution for traceability between various product fragments through the lifetime of the system does not exist.

Central repository for the traceability relationship and distributed repositories for the model fragments storage and interchange between developers are proposed. Usage of ontology is proposed to interrelate different product fragments and establish the traceability relations between them.

1. Introduction

Traceability and its relevance to the software development process are well described in the literature (e.g., [8], [28]). Especially much research is done in pre-requirements traceability to capture rationale (e.g., [9], [22], [24]), but there is the lack of traceability through the entire system development process.

An information system is viewed as a product composed of product fragments, which are again compositions of sub-fragments:

– *Model fragments* – are sub-models of a conceptual model of the information system being under development. Only the semantic content of the model is stored, not diagram layout information.

– *Diagrams* – stores layout information of the conceptual model view. Diagrams may exist in several different layout versions without affecting the conceptual content.

– *Code fragments* – are code modules (files).

– *Document fragments* – are pieces of the documentation of the models and code fragments.

These four fragments types are described differently. Model structure depends on the conceptual model languages used, diagram structure reflects the basics of visual languages, code structure is expressed in programming language and document structure reflects common document architectures.

Every decision and rationale behind it should be captured and traced during system development. The traceability of the history of a product is a prerequisite for managing evolution of product. Capturing and maintaining traces from requirements to implementation and vice versa have long ago been acknowledged as one of essential systems development activities [19].

There are several aspects that make the traceability between requirements and later fragments of software development (design, architecture, code fragments) problematic. First, conceptual distance between two worlds: human (requirements are captured in natural language) and technical (entities are specified in formal method). Second, it is difficult to maintain the consistency and traceability between different fragments since single requirement could map multiple architectural and design concerns derived from it. Contrarily, architectural component could have few relations to various requirements. And third, large systems should satisfy hundreds, even thousands of requirements, and this makes second issue even more complicated.

This paper is the statement about research in progress within the area of traceability between various fragments (documents, models, model elements, code) in a collaborative software development throughout entire system lifecycle. The research has so far mainly been built on an extensive literature study and survey of the requirements management tools.

The paper is organized as follows: first, the traceability issues and existing approaches are discussed in light of system development lifecycle. Next section discusses the aspects of traceability in a collaborative work and lists deficiencies of CASE tools. The research issues and a vision of approach to tackle them are enlightened in section 4. Finally, conclusions are presented.

2. Traceability from a Life Cycle Perspective

Gotel and Finkelstein [8] define requirements traceability as “the ability to describe and follow the life of a requirement in both forwards and backwards direction (i.e., from its origins, through its development and specification, to its subsequent deployment and use, and through all periods of on-going refinement and iteration in any of these phases)”.

Software systems always evolve as the environment and stakeholders’ requirements change. Therefore, managing change is a fundamental activity not only in requirements engineering (RE), but also in overall system development.

Neglecting traceability or capturing insufficient traces could decrease system quality, and extend development time. It is important to trace which model element is affected by change of requirement. And, vice versa, how change of model element corresponds to defined requirements, and consequently how will the system functionality be affected.

The systems specification is developed in an iterating manner. Concurrent Engineering requires the cooperation of people coming from different phases of the engineering process. Traceability between the different views (diagrams), which exist in such cross-functional teams, is essential for enabling mutual understanding. Moreover, the different views must be related to each other and must be presented in a suitable way to support finding and resolving of inconsistencies, conflicts, and different opportunities.

One of the solutions to the problem of traceability is the hyper linkage of documents from different phases of development. If these documents are suitably annotated, they can provide a meaningful design history throughout the development lifecycle [3] and increase the browsing capabilities. This could be done mainly manually or semi-automatically using linguistics technique. Cerbah and Euzenat [2] have implemented system that generates class hierarchies out of textual requirements specifications and establishes traceability between models and texts through terminology. The authors have not explicitly stated, but the same technique could be use to relate the documentation.

The CBSP (Component, Bus, System, Property) approach [11] deals with refinement of requirements to initial architecture, as requirements may explicitly or implicitly contain information relevant to the system’s architecture. A Scenario and Meta-Model Based Approach [23] integrates requirements and architecture information by defining orthogonal meta-models, which define the concepts and relations about information to be recorded during system development. It is based on six meta-models. Meta-models are used to structure requirements information and interrelate structured information.

Murphy et al. presents formal technique for abstraction of source code information to match higher-level model elements [17]. The abstractions for consistency checking between model and code, but not interrelation between different model elements, are used.

Approach by Hall et al. [12] uses extended problem frames [14], which allow architectural structures, services and artefacts to be considered as part of the problem domain.

Jviews by Grundy et al. [10] uses class-like diagrams and other design-level constructs. They focus on lower-level design and address a series of view integration problems in that level.

ViewPoints [6] framework by Finkelstein et al. presents some views and corresponding rules to identify inconsistencies within and between them. Framework provides mechanisms for detecting, classifying and resolving inconsistencies. ViewPoint resorts to a more formal specification of requirements. ViewPoint support for design and architecture type diagrams is not explicitly stated, although they state that some exists.

Comparison of those approaches ([2], [6], [10], [11], [12], [17], [23]) for relating different views in system development is summarized in table 1. Approaches are compared in context of their coverage of lifecycle phases. Lifecycle phases are split into requirements, architecture, design, coding and system documentation. System development lifecycle is assumed to be an iterative process, not waterfall-like.

Table 1. Comparison of approaches from lifecycle perspective

Phases of lifecycle / **Aproaches**	Requirements	Architecture	Design	Coding	Documentation
Linguistics approach [2]	■	□	◩	□	■
CBSP [11]	■	■	□	□	□
Scenario & Meta-Model Based [23]	■	■	□	□	□
Murphy et al. [17]	□	□	◩	■	□
Approach using Problem Frames [12]	■	■	□	□	□
Jviews [10]	□	□	■	□	□
ViewPoints [6]	■	◩	◩	□	□

■ - high ◩ - average □ - none

The following ratings were assigned: *high*, *average*, *none*. The average and *high* ratings are given when some support or extensive support is available. A *none* rating is assigned when the criteria are not satisfied by the approach or are insignificant.

In general, research is done in the area of traceability between two different phases of software development (either between requirements and rationale behind them, or requirements and architecture, or source code and model elements (see table 1)). Most of the investigated approaches do not take into consideration documentation phase of system lifecycle (user manuals, blueprints and etc.). Consistent update of the system documentation is one of the prerequisites for its maintainability. Of course, documentation could be considered to be similar to requirements specification documents (as content is mainly in natural language). Nevertheless, most of approaches at requirements level deals with semi-formal specification (scenarios, use case) and they hardly consider plain language documents. This raises the first question: is it possible to maintain backward and forward traceability between various fragments through the lifetime of the system?

Desirable types of traceability relationships in an iterative system development are depicted in fig. 1. Interrelationships between related fragments of different abstraction levels inside of every stage should be maintained as well as relationships between fragments of different lifecycle stages. It is important to control more detail representation level conformity to the problem domain, as abstraction mechanisms are used to simplify picture of the system and problem domain, which is further used for communication with the non-technical users.

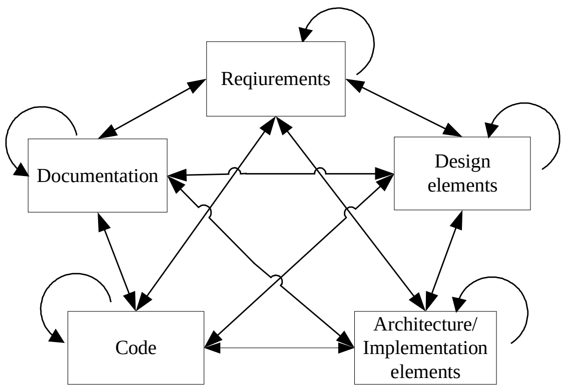


Figure 1. Desirable traceability links throughout system lifecycle.

The necessity of the link between requirements and code is not so obvious, but could be useful in an evolutionary prototyping, when a prototype is used to extract the requirements from IKIWISI (I'll know it when I see it) users.

3. Traceability in Environment for Collaborative System Development

3.1. Deficiencies of the CASE tools

Large, net-wide collaborative system engineering projects become a reality by the recent growing communication and data exchange possibilities brought about by the fast-growing Internet-related developments. These projects share system models among users, tools and repositories beyond geographical boundaries, which could result in significant savings in time and effort. Models that have been created or processed by one tool could serve as the initial step for the next tool. Shared models could also enable the composition and integration of the most appropriate tools for a given activity. Especially, as many CASE (Computer Aided System Engineering) tools are developed as “time sharing” systems [5]: each developer is given the feeling of being the “only user” of the system.

In addition, the format of the product objects are often controlled by strict consistency checks, making the evolution of product objects from informal ideas to formal constructs difficult. As a consequence, CASE tools might be reduced to tools for documenting products that are developed outside the CASE tools. That is not what CASE tools are built for. CASE tools in the best case confine themselves to offering a central repository where information about the product can be accessed regardless of geographical location.

Weaknesses in CASE tool support could be divided into the following aspects [20]:

- Lack of mechanism for integrating sets of methods while maintaining consistency between various models,
- Lack of support of multiple users to create, modify and delete sets of partly overlapping model instances,
- Inadequate catering for multiple representational requirements raging from fully diagrammatic to fully textual or matrix representation.
- Failure to provide consistent mapping mechanism between different representational paradigms.
- Lack of flexibility and evolvability in method support ranging from syntactic variation in methods to crafting totally new method components.
- Insufficient catering for different information-related needs of a diverse set of stakeholders.

The cost-efficient sharing of modelling information between heterogeneous tools and repositories requires the adoption of a standard for an industry-wide model interchange format. However, no such standard yet exists. There are several model interchange formats: XIF [16], XMI [21], SPOOL [26], UXF [27]. Product fragments are stored and interchanged through a repository. The repository may be useful during various phases in the lifecycle of an information system (for instance, as a basis for various decisions or reuse).

3.2. Specific Aspects of Collaborative Environment

Each engineer develops its own product (fragment) using his/ her preferred representation. At a certain time, the products developed in parallel must be integrated; discrepancies and similarities must be detected through the communication and conversation among the people involved. Changes to the products have to be made according to unresolved discrepancies. Fundamentally, concurrent engineering relies on the capability to merge pieces of work done in a concurrent way on the same object. Merging objects is thus a central issue. Trace information should be captured and traceability between related fragments should be re-established.

Farshchian in [5] emphasizes the list of requirements for product development environments; some of them (relevant to this research) are listed below:

- Flexible access to the product – a product development environment should provide flexible mechanisms for accessing and updating the product.
- Unrestricted product object types – a product development environment should allow the developers to share any type of object that they might find useful for supporting their cooperation.
- Unrestricted relation types – a product development environment should allow the developers to create any type of relation between any two objects of product.
- Incremental product refinement – a product development environment should provide the developers with flexible mechanisms for incrementally refining the product. The developers should be allowed to start with vague products, and to refine them into more complete and formal ones.
- Support for boundary objects – a product development environment should allow the developers to view the product from different perspectives. The environment should in addition support a global view of the product.
- Active delivery of information – a product development environment should take an active part in delivering necessary information to the developers. In particular information about changes to the shared product should be delivered continuously to the interested developers.

Traceability technique integrated with the version control and configuration management could facilitate management of the composition of product fragments consisting of interrelated various model fragments, code fragments and documents:

Merging. Merging allows multiple versions to be joined together, producing a new version representing the union of the actions taken from previous version. Fully automatic mergers are difficult to implement due to semantic considerations during the resolution of conflicting changes. In case the change has been applied in only one version, this change can be incorporated automatically; otherwise, a conflict that can be resolved automatically or manually is detected. Merging should handle various levels of granularity.

Access control should be applied at different level of granularity, for instance, model, object, and attributes.

Version management. The collaborative CASE tools should keep track of changes in different working modes [15]:

- Multiple developers are working on a single common version simultaneously;
- Developers are working individually on their local versions;
- Both cases: some developers collaborating synchronously, others working individually.

In first case the system should track and resolve multiple edits by different developers in the same fragment. Changes could occur simultaneously or sequentially in collaborative session. Considering second case – there will be a number of current versions – the system should provide and manage awareness of their existence and dependencies among them. Third case incorporates previous both – asynchronous and synchronous developing.

In [13] it is stated five different properties of configurations must be maintained during systems development:

Authorized. A configuration is authorized if and only if all of its objects are defined as being authorized. An authorized configuration is also called a baseline.

Consistent. The objects of a consistent configuration have to be consistent, both relative to each other and as individual components. A configuration remains invalid if it is not checked or if the consistency checker detects inconsistencies.

Latest. Each object of a latest configuration must be the latest version of its own family.

Owner. The property owner constrains the configuration to contain only object versions owned by a given user. Authorized versions are considered as owned by all project group members.

Project-wide. A project-wide configuration must include one object version from every family within project.

Three configurations are important: 1) the *latest project-wide*; 2) *latest consistent*; 3) *latest authorized*. The latest project-wide baseline reflects all new developments in the project. Latest baselines are not supposed to be consistent, but all new ideas since last logon are detected by inspecting this configuration. Latest consistent baselines represent the most recent stable work. These configurations are the candidates for authorization. There may exist several latest, authorized configurations, but usually exists one. This is the last configuration, which the group agree on, and it forms the official basis of all subsequent work. Traceability technique between different versions of the product fragments with incorporated configuration management facilitates extraction of relevant configuration.

4. Research Issues and Future Work

From a development perspective four particularly important areas have been identified: traceability from requirements to implementation elements through design, interchange of product fragments, configuration management and concurrent work.

Internet technologies (XML, RDF) contribute the model interchange formats for trace information by capturing and exchanging through repositories, and enabling semantic interoperability. Model interchange formats should support any inter-model consistency check, and semantic validation. They should not be used only for static model transfer between environments.

4.1. Research areas

Considering perspectives mentioned in the chapters above, two main types of traceability should be maintained in an ideal cooperative software development:

- Traceability from requirements to implementation – most difficult to implement as various diagrams could be used in the development project;
- Traceability between versions and configuration. The information about the essence of new version development and the rationale behind it should be captured. This is similar to the pre-traceability task – to capture the rationale behind requirements.

The question is: *What kind of trace information is possible to capture?* It could be answered over the time when understanding of the domain and solution increases, as currently there is impossible to capture all trace information.

While work in software architectures has concentrated on how to express software architectures and reason about their behavioural properties, there is still an open question about how to analyse what impact a particular architectural choice has on the ability to satisfy current and future requirements, and variations in requirements across a product family [7].

The main research question is:

How can we effectively and consistently integrate changes of development objects in different levels of granularity throughout lifecycle of geographically distributed cooperative product development?

4.2. Future Work

Concurrent engineering changes old practice, when all the required objects were locked during the whole change/ modification activity. Each software engineer should have direct access to all needed objects. But changed version should be kept with access forbidden for other developers during modification, because the state of fragment is inconsistent in a modification phase. If *n* engineers change the same object concurrently, this object should have $n+1$ different copies [4]. It means that each developer needs the private copies of fragments. On the other hand, the colleagues know that other changes possibly are done on the same fragments/ objects and want to be incorporated when relevant. During the development all relevant information should be captured, traced and available for all project participants, depending on access rights.

The requirement for consistent evolving product structures introduces high complexity in the software configuration management systems. Furthermore, this requirement restricts cooperative work on the same structure since cooperative work necessarily means that the product structure is in a state of inconsistency most of the time [1]. “Lazy” consistency [18] could be introduced to avoid disturbance of the developers’ creativity and to satisfy the requirements for the *incremental product refinement*. This approach favours software development architectures where impending or proposed changes, as well as changes that have already occurred, are announced. This allows the consistency requirements of a system to be “lazily” maintained as it evolves. Lazy consistency maintenance supports activities such as negotiation and other organizational protocols that support the resolution of conflicts and collisions of changes made by different developers.

In figure 2 an architectural proposal for traceability management in distributed cooperative system development is depicted. The central repository is used for storage of traceability links; the model fragments are stored on local workstations in shared repositories. Group awareness and access control techniques are to be used to manage collaborative work of developers. Version control and configuration management techniques are to be used on the common repository side for the traceability relations’ version control and update. These techniques are also to be used on the workstations side for the configuration management of product fragments.

Traceability relations should be based on semantic interoperability of related fragments. Every fragment has knowledge of the developer that s/he represented in. Semantic is the relationship between the fragment and the meaning (knowledge) it possesses. In order to establish the relation between the fragments, an ontology of what the fragments are about should be defined. So, the repository for traceability relations (depicted in fig. 2)

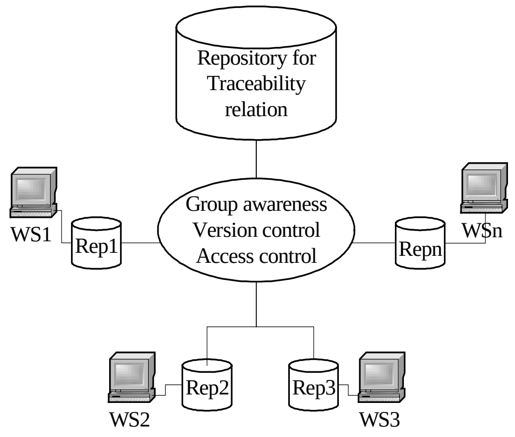


Figure 2. Deployment of traceability management in collaborative development.

should have meta-model as pre-defined ontology. Ontology would facilitate the usage of *unrestricted* product objects and relation types.

5. Conclusions

A transition between various product fragments is still unsolved problem, despite of several methods for linking requirements to architecture components (e.g., [11], [23]).

There are very few empirical studies focused on how organizations actually manage the different types of traceability. More research is required on how organizations actually capture the trace information. The real challenge is not only to investigate and propose solutions for those problems, but it is rather to provide solutions, which outweigh the cost of implementing and using them. Because the manual mapping and hyper linkage establishing are not the solutions that could be enthusiastically accepted by industry, product fragments (natural language, models and code) should be woven into a coherent whole.

It is not trivial to generate and validate trace information, but traceability could help the company:

- To ensure completeness: The user or a program can easily identify the requirements which are not satisfied by the system by following traceability links;
- To propagate the changes: At any time in the development process, traceability information allows to find out the elements impacted by changes. For instance, it could be possible to evaluate the impact on the software design and implementation of a late change in the initial customer requirements.

Phases in system development and their output are so different (one of the reasons – various modelling techniques are used) that to develop multipurpose traceability method is hardly possible. By now it is possible to develop solutions for separate modelling languages as UML [25], PPP [29] by defining the common ontology for the semantic interrelation of fragments.

6. References

[1] R. Andersen, *A Configuration Management Approach for Supporting Cooperative Information System Development*, PhD thesis, NTH, Trondheim, Norway, 1994

[2] F. Cerbah, J. Euzenat, “Traceability between models and texts through terminology”, *Data and Knowledge Engineering,* Elsevier Science Publishers, 2001 38 (1), pp. 31-43.

[3] J. Corriveau, C. Hayashi, “A Strategy for Realizing Traceability in an Object-Oriented Design Environment”, *Proceedings of Computer Aided Systems Theory - CAST'94,* 4th International Workshop, Ottawa, Canada, May 1994.

[4] J. Estublier, “Objects Control for Software Configuration Management”, In K.R. Dittrich, A. Geppert, M.C. Norrie (Eds.): *Advanced Information Systems Engineering, Proceedings of thirteenth conference CAiSE 2001,* LNCS 2068, Springer-Verlag, Interlaken, Switzerland, June 2001, pp. 359-373.

[5] B.A. Farshchian, *A Framework for Supporting Shared Interaction in Distributed Product Development Projects*, PhD thesis, NTNU, Trondheim, Norway, 2001.

[6] A. Finkelstein, J. Kramer, B. Nuseibeh, L. Finkelstein, M. Goedicke, “Viewpoints: A Framework for Integrating Multiple Perspectives in System Development”, *International Journal on Software Engineering and Knowledge Engineering,* World Scientific Publishing, march 1991, pp. 31-58.

[7] D. Garlan, “Software Architecture: A Roadmap”, In A. Finkelstein (ed.), The Future of Software Engineering, Special Issue 22nd International Conference on Software Engineering, ACM Press (2000)

[8] O.C.Z. Gotel, A.C.W. Finkelstein, “An Analysis of the Requirements Traceability Problem”, In *Proceeding of the 1st International Conference on Requirements Engineering* (ICRE’94), IEEE Computer Society Press, Colorado Springs, Colorado, USA, April 1994, pp. 94–102.

[9] O.C.Z. Gotel, A.C.W. Finkelstein, “Contribution Structures”, In *Proceedings of the 2nd IEEE International Symposium on Requirements Engineering* (RE'95), IEEE Computer Society Press, York, U.K., March 1995, pp. 100-107.

[10] J. Grundy, J. Hosking, W. Mugridge, “Supporting flexible consistency management via discrete change description propagation”, *Software Practice and Experience*, John Wiley & Sons, 1996 26(9), pp. 1053-1083.

[11] P. Grünbacher, A. Egyed, and N. Medvidovic, “Reconciling Software Requirements and Architectures - The CBSP Approach”, In *Proceedings of the 5th IEEE International*

Symposium on Requirements Engineering (RE'01), Springer-Verlag, Toronto, Canada, 2001, pp. 202-211.

[12] J.G. Hall, M. Jackson, R.C. Laney, B.A. Nuseibeh, L. Rapanotti, "Relating Software Requirements and Architectures using Problem Frames", (to appear in) *Proceedings of IEEE International Requirements Engineering Conference* (RE'02), Essen, Germany, September 2002.

[13] T.R. Henriksen, A.D. Fidjestøl, A.B. Aubert, "PPP Repository Management", *Unpublished NTNU report*, Trondheim, Norway, 15th October 1997

[14] M. Jackson, *Problem Frames*, ACM Press Book, Addison-Wesley, 2001.

[15] B.G. Lee, N.H. Narayanan, K.H. Chang, "An integrated approach to distributed version management and role-based access control in computer supported collaborative writing", *The Journal of System and Software*, Elsevier Science Publishers, 2001 59 (2), pp.119-134.

[16] Microsoft Corp., "Repository SDK 2.1b Documentation: XML Interchange Format (XIF)", Microsoft Corp., Redmond, USA, May 1999.

[17] G.C. Murphy, D. Notkin, K. Sullivan, "Software Reflexion Models: Bridging the Gap Between Source and High-Level Models", In *Proceedings of the 3rd ACM SIGSOFT Symposium on the Foundations of Software Engineering*, New York, NY, 1995, pp.18-28.

[18] K. Narayanaswamy, N. Goldman, ""Lazy" Consistency: A Basis for Cooperative Software Development", In *Proceedings of International Conference on Computer-Supported Cooperative Work* (CSCW'92), Toronto, Ontario, Canada, November 1992, pp.257-264.

[19] B. Nuseibeh, S. Easterbrook, "Requirements Engineering: A Roadmap" In A. Finkelstein (ed.), *The Future of Software Engineering*, Special Issue 22nd International Conference on Software Engineering, ACM Press, 2000.

[20] S. Kelly, K. Lyytinen, M. Rossi, "MetaEdit+ A Fully Configurable Multi-User and Multi-Tool CASE and CAME Environment", In *Proceedings of CAiSE'96*, Heraklion, Greece, May 1996.

[21] OMG, "XML Metadata Interchange (XMI)", Document ad/98-10-05, October 1998.

[22] F.A.C. Pinheiro, J.A. Goguen, "An Object-Oriented Tool for Tracing Requirements", *IEEE Software*, 1996 13(2), pp. 52-64.

[23] K. Pohl, M. Brandenburg, A. Gülich, "Integrating Requirement and Architecture Information: A Scenario and Meta-Model Based Approach", In *Proceedings of the Seventh International Workshop on Requirements Engineering: Foundation for Software Quality* (REFSQ'01), Interlaken, Switzerland, 2001.

[24] K. Pohl, "PRO-ART: Enabling Requirements Pre-Traceability", In *Proceedings of the Second International Conference on Requirements Engineering* (ICSE '96), Colorado, USA, 1996, pp. 76-85.

[25] Rational Software Corp., "Complete UML 1.4 specification", 2001.

[26] G. St-Denis, R. Schauer, R.K. Keller, "Selecting a Model Interchange Format The SPOOL Case Study", In *IEEE Proceedings of the 33rd Annual Hawaii International Conference On System Sciences*, Maui, Hawaii, 2000.

[27] J. Suzuki, Y. Yamamoto, "Managing the Software Design Documents with XML", In *Proceedings of the Sixteenth Annual International Conference of Computer Documentation* (ACM SIGDOC '98), Quebec City, Canada, 1998, pp. 127-136.

[28] R. Watkins, M. Neal, "Why and How of Requirements Tracing", *IEEE Software*, 1994 11(4), pp. 104-106.

[29] M. Yang, *COMIS – A Conceptual Model for Information Systems*, PhD thesis, NTH, Trondheim, Norway, 1993.

F.2 A Vision for Product Traceability based on Semantics of Artifacts

Strasunskas, D. A Vision for Product Traceability based on Semantics of Artifacts. In Al-Ani, B., Arabnia, H.R., and Mun, Y. (Eds.) *Proc. of the 2003 Intl. Conf. on Software Engineering Research and Practice* (SERP'2003), part of Intl. MultiConference in Computer Science & Engineering, CSREA Press, Vol.II, ISBN:1-932415-20-3, Las Vegas, Nevada, USA, June 2003, pages 890-895.

A Vision for Product Traceability based on Semantics of Artifacts

Darijus Strašunskas
Dept. of Computer and Information Science, Norwegian Univ. of Science and Technology
Sem Sælands vei 7-9, NO-7491 Trondheim, Norway

Abstract

In the face of extensive attention form both the research community and the industry, traceability there still lacks of a supporting methodology that enables traceability throughout the whole lifecycle of a system. In particular, attention need to be given to geographically distributed development efforts where developers are likely to use different representation formats and a variety of tools for the product development.

An approach to methodological support for artifact management and traceability is presented in this paper. Fragments from different development phases (i.e., requirements specification, design, code, test scenarios, and documentation) are linked to concepts from a domain model and further, interlinked through it. A conceptual domain model is constructed from domain specific concepts (nodes) and quantified relationships between them. An initial domain model and the weights for concept relations are based on the experts' experience and knowledge from previous projects. The main contribution of this work is two fold. First, the approach covers the whole product traceability. Second, prediction and assessment of impact are enabled by tracing related fragments through relations of concepts in a domain model.

Keywords: product traceability, distributed collaborative development, semantic enrichment

1. Introduction

Information system development is a highly iterative process, in which developers try to capture the needs and desires of all stakeholders, and transform them into a complete system, consisting of both manual and computerized parts. The product of such development projects undergoes changes because of their iterative nature. Traceability is defined as a property of a system description technique that allows changes in one of the system descriptions – requirements specification, design, code, documentation, or test scenarios – to be traced to the corresponding fragments of the other descriptions [6]. Such correspondence relationships should be maintained throughout the life time of a system in order to manage the artifact.

Traceability and its relevance to systems development have received much attention in the requirements engineering literature (e.g. [5][18]). Especially, the pre-requirements traceability has been studied severely (e.g. [11][12]). However there is a lack of traceability tools to support the full life-cycle, starting from artifact inception to its use. Different representation formats that are used throughout the development process make it complicated to cover the whole life-cycle of an artifact. Given that a single requirement map to multiple architectural and design concerns which are derived from it, it is difficult to maintain the consistency and traceability. Moreover, an architectural component has a number of relations to various requirements. The task becomes even more difficult in the face of a large system that is build to satisfy thousands of requirements.

In a geographically distributed project developers may use different tools to create and modify product fragments, which can be refined iteratively and further processed by colleagues. Afterwards produced fragments are interchanged among members of a project, so that is important for colleagues to interpret an artifact correctly. These are the main challenges for traceability - to interrelate and trace all artifacts in different representation formats that are produced in a distributed manner using different tools and to cover the whole product lifecycle.

An approach to product fragments management and traceability during the distributed collaborative development process is presented in this paper. Artifacts mapping to the corresponding concepts from a specific problem domain increase the semantics of an artifact. Having the interrelated concepts from the problem domain and all fragments linked to them, it is possible to predict and assess fragment change impact on other fragments.

The overall structure of the paper is as follows. In section two, related works are analyzed. In section three, proposed approach for product traceability is presented. In section four, possible implementations of proposed methodological approach are listed. Finally, in section five, the work is concluded and possible shortcomings and insights how to solve them are discussed.

2. Related Work

Traceability issues have been tackled and a number of techniques have been proposed for providing traceability, such as [5]: cross referencing schemes, based on some form of tagging, numbering, or indexing; requirements traceability matrices. Studies done in the field of traceability have mainly focused on specific parts of the development process [16] – mostly in the areas of pre-requirements traceability (e.g. [11][12]) and linking requirements to architectural components (e.g. [7], [13]).

There are approaches based on specific modeling language and /or tool. A much cited tool is TOOR (Traceability of Object-Oriented Requirements), presented by Pinheiro and Goguen in [11], it is based on FOOPS, a formal object-oriented language. Letelier [9] presents a framework for configuring requirements traceability by integrating textual specifications and UML (Unified Modeling Language) model elements. Proposed approach is restricted to UML language and can be applied to software process based on UML.

Some approaches deal with establishing traceability links thereafter the most of a system is developed (e.g. requirements specification, code) and, per se, contribute mainly for product maintenance. Frezza et al [4] propose a system of simulation where both the requirements and implemented system are simulated in order to obtain a set of result data. The data from the requirements and implementation are then compared, which result in a quantitative measure of how accurate the running system implements the requirements. Egyed [2] suggests using a scenario driven approach to acquire runtime information about a system and relate the information – footprints - to the requirements and model of the running system. The footprints are then analyzed in a tool *Trace Analyzer*, which shows how the components of the system interact when performing specified scenarios. Thus, it is possible to obtain added trace information on how the running system actually fulfills its requirements and which parts of the design are affected.

Ramesh and Jarke in [14] offer a wide vision about the information needed in requirements traceability. Their study is based on the analysis of industrial software development projects. They identify two segments of traceability users and suggest two corresponding traceability meta-models (one is a simplification of the other). In this work the only suggested mechanism to configure the meta-model according to the project needs is to cut or to add parts of the meta-model.

Hence, in general, there is a lack of support and coverage of whole product lifecycle. There is also noticeable disregard of support for distributed teams using different tools and representation techniques and notations. Of course, there are development environments (e.g. Rationl Suite AnalystStudio [15]), which compound together programs for requirements engineering, design, change management and code repository. Though most of integrated programs do not support collaborative work; not all project phases are equally well supported and, by choosing this kind of tool environment, customer is bound to one vendor.

3. Proposed approach - Mapping to the Domain Concept

The objective of this approach is to enable change notification and impact prediction through all phases of development in the distributed projects. That means that different tools and, most likely, different notations will be used during the project. In [3] the list of requirements for product development environments to enable collaboration in geographically distributed developments is emphasized; some of them (relevant to this research) are listed below:

– Unrestricted product object types – a product development environment should allow the developers to share any type of object that they might find useful for supporting their cooperation.
– Unrestricted relation types – a product development environment should allow the developers to create any type of relation between any two objects of product.
– Incremental product refinement – a product development environment should provide the developers with flexible mechanisms for incrementally refining the product. The developers should be allowed to start with vague products, and to refine them into more complete and formal ones.

The traceability approach is based on the requirements to support for collaboration in distributed projects as listed above. There are two basic assumptions underlying as follows:

– CASE-tools (Computer Aided Software Engineering) that are used during the product development support XML (eXtensible Markup Language) or XML-dialect format output of developed fragments. The assumption is reasonable, since most CASE-tools maintain model interchange formats derived from XML.
– There is a problem domain and it can be characterized by well-defined, interrelated concepts. Furthermore these concepts are represented as entities having weighted relationships which show the strength of relationship between the concepts. This assumption is more restrictive since not all entities/relationships can be assigned weights.

Domain model is constructed and all concepts are connected with weighted links according how strongly concepts relate. Those weights further are used to evaluate interrelations between fragments mapped to the domain concepts and to estimate likelihood of impact of one fragment to another. Traceability relations are based on the semantics of the artifacts. Fragments are linked to the concepts from the domain model; all fragments are mapped and linked through the conceptual domain model as follows. There exists a domain model such that:

– If fragment F_i is linked to a concept C_A and fragment F_j is linked to C_A, then transitively F_i also relates to F_j:

$$\mathbf{F_i} \to \mathbf{C_i} \wedge \mathbf{C_i} \to \mathbf{F_j} \Rightarrow \mathbf{F_i} \to \mathbf{F_j}$$

– Having related concepts C_A and C_B, and if fragment F_i is linked to a concept C_A and a fragment F_j is linked to C_B, then trace dependency *in some degree* exists between F_i and F_j.

$$\mathbf{C_i} \to \mathbf{C_j} \wedge \mathbf{F_i} \to \mathbf{C_i} \wedge \mathbf{C_j} \to \mathbf{F_j} \Rightarrow \mathbf{F_i} \to \mathbf{F_j}$$

Meta-model for the proposed approach, based on settings described above is depicted in figure 3 using RML (Referent Model Language) [17]. A product is final result of the development project, and it consists of the interrelated products of phase. Product of phase is used to relate specific phase of lifecycle to artifacts developed within the *phase* (e.g., business analysis, requirements engineering, design, implementation, testing and etc.). Phase products are related by has_change_impact_to relation in order to restrict change notification and propagation only to adjacent phase products. Consider that, developers are notified only about possible impact on fragments from the “surrounding phases”. For example, if a piece of code has been changed, the developers first need to check it whether there is some impact on design. Further if design fragment is impacted, then trace back to requirements. Finally, if no impact is present, - change notification because of that change of code stops. It should be noted that relations are not based on sequence of phases in a lifecycle (product development lifecycle is not assumed to be waterfall-like), because a lifecycle usually is highly iterative where phases could be repeated and concurrent where several phases could be developed at the same time. Phase products are related to each other according to the logical dependence between the content of the phase products (e.g., requirements specification and test scenarios).

Stakeholders are responsible of creating and modifying the fragments. A fragment is a semantic piece of phase product in a certain granularity level, e.g., it can be a document, a model, a diagram, a section in a document, a text specifying a non-functional requirement, an use case, a class, an attribute, etc. Links from one fragment to another denote direct dependence between fragments and should be established when possible. Every fragment has semantics, which relate the fragment to one or more domain concept cluster. Weighted mapping relationships are used to distinguish fragment coherency to a particular concept. Domain model is composed from domain concepts. Domain cluster can consist of one or more concepts. Domain concepts are connected by as direct acyclic graphs with weights (weighted relationships). Weights of those relations are calculated based on degree of the concept relatedness.

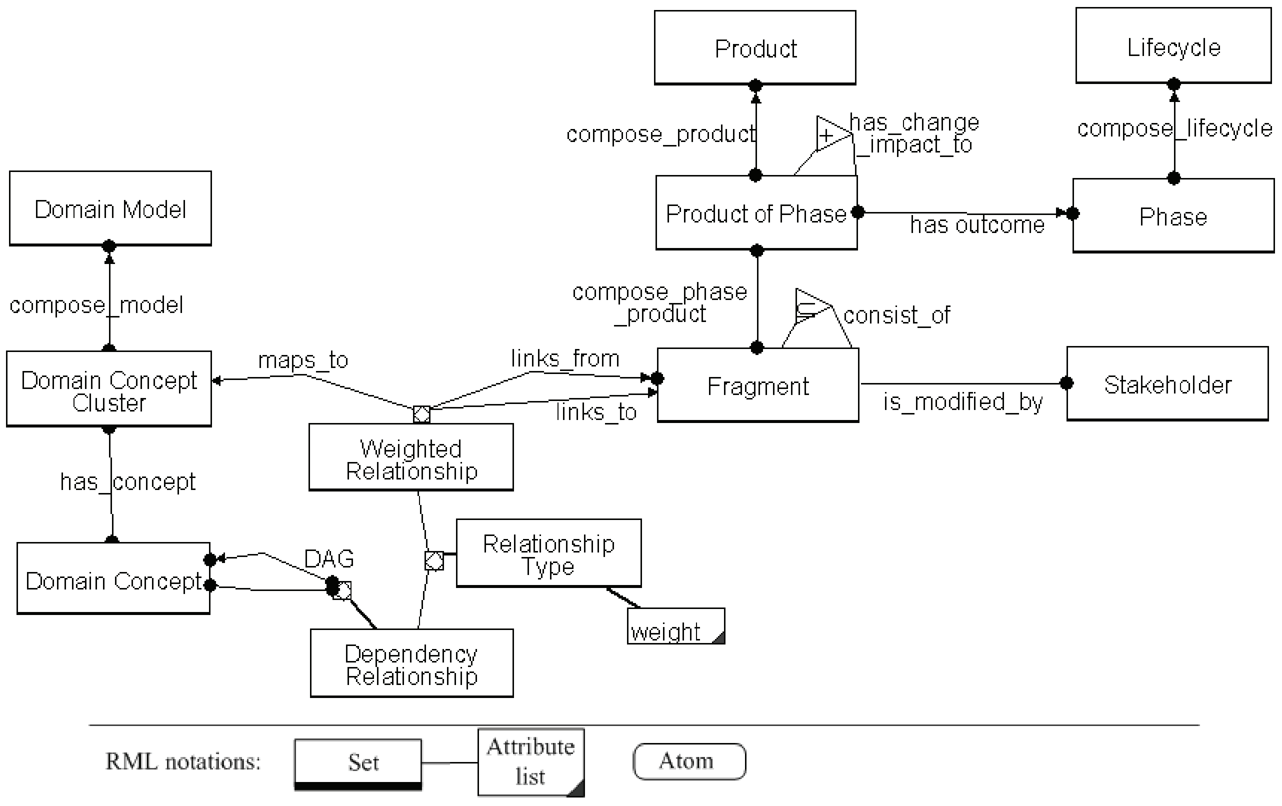


Figure 1. Product traceability meta-model

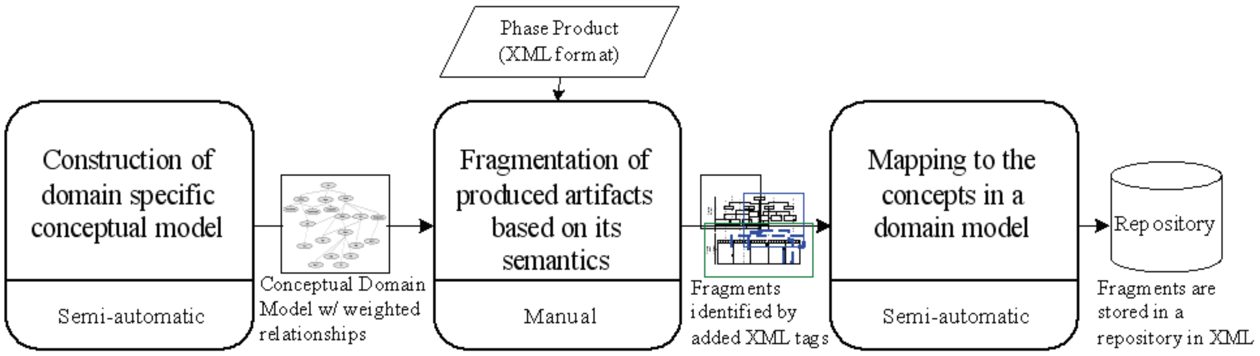

Phase Product
(XML format)
Construction of domain specific conceptual model
Semi-automatic
Conceptual Domain Model w/ weighted relationships
Fragmentation of produced artifacts based on its semantics
Manual
Fragments identified by added XML tags
Mapping to the concepts in a domain model
Semi-automatic
Repository
Fragments are stored in a repository in XML

Figure 2. Main steps to enable product traceability

The basic steps in approach are (see fig.2):

1. ***Building a conceptual domain specific model.*** This step consists of two main sub-steps: *(a)* extraction of domain specific concept and *(b)* weighing of relationships between concepts (see fig.3).

a) Syntactical analysis of textual documents has been investigated severely in last few decades. Natural language processing is a main technique used to extract more structural information out of documents. Efforts are directed to build models from requirements specification in natural language. The naïve approach is to use nouns as candidates for entities and verbs for relations between entities. However, there is necessity for more sophisticated techniques to handle linguistic variation when proposing model elements when constructing domain models from a large set of documents. [1] proposes approach of natural language analysis for semantic documents modeling, where techniques for domain model construction are discussed.

b) Quantification of the relationship between concepts could be done using linguistics and natural language processing techniques for analyzing the documents from a domain. Collocation technique and text mining are used to evaluate the strength of relationship between concepts. The values should be refined by the domain expert – this reflects domain expert's belief in how much concepts are related in a particular domain. So, these numbers come from either objective data or the experiences of the domain expert accumulated from the development of similar projects. These ranges can be used to represent the high (0.7 to1.0), medium (0.4 to 0.6) and low (0.0 to 0.3) degree of relation.

2. ***Fragmentation of artifacts into semantic fragments***. Produced artifact is translated to XML format and logically fragmented according its semantics. Fragmentation is done by a traceability module which gets the XML file as input and developer defines fragment boundaries. As an output, XML file with added tags to identify start and end positions of fragment is produced.

3. ***Fragments mapping to the concepts.*** Candidate concepts form domain model are suggested automatically by processing the fragments. Techniques from first step are adapted to extract concepts, if possible, from the fragments and propose the closest related concept from domain model to map to it. Fragments can be linked directly to other fragments if developer finds them related or one fragment is part of another (recall fig.1). Also fragment can be manually linked to domain concept. The weighing scheme is used the same as described in step 1b). Relation information is encoded by XML tags. Finally fragments are stored in a central repository.

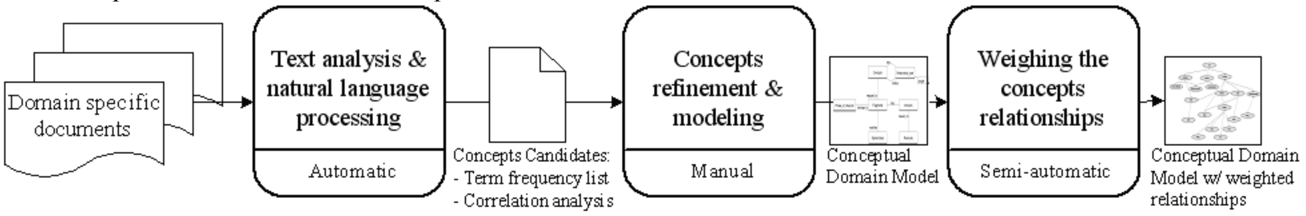

Domain specific documents
Text analysis & natural language processing
Automatic
Concepts Candidates:
- Term frequency list
- Correlation analysis
Concepts refinement & modeling
Manual
Conceptual Domain Model
Weighing the concepts relationships
Semi-automatic
Conceptual Domain Model w/ weighted relationships

Figure 3. Process for construction of domain specific conceptual model

4. Application of the approach

Good candidates for implementing the approach for product traceability are Bayesian Belief Network (BBN, also called Bayesian Network or Probabilistic Networks) or weighted graphs. BBN is a powerful technique for reasoning under uncertainty [8][10] and representing knowledge. It provides a graphical model that resembles human reasoning. In last decades Bayesian Belief Network has attracted much attention from both research and industry communities. BBN provides a natural way to structure information about a domain, resembling human reasoning. One advantage of the BBN is that it can not only capture the qualitative relationships among variables (denoted by nodes) but also quantify the conceptual relationships. This is done by assigning conditional probability to each node in the BBN.

Weighted graphs could be used to represent a conceptual domain model. Interrelation of the concepts could be depicted as a distance between concepts. The shortest path algorithm could be used to predict which fragments could be impacted.

5. Discussion and Conclusions

Proposed approach *(a)* enables full lifecycle product traceability. As nature of collaborative development is usually very iterative, the approach *(b)* allows tracing and interchanging product fragments at different stages of its incremental refinement (e.g., from abstract sketches to formal representation), *(c)* does not bind developers to a specific tool and/or modeling language, as far as used tool supports XML output. The use of XML makes it possible to use this approach in settings where the involved artifacts are created and managed by heterogeneous tools, such as text processors and CASE-tools.

Proposal can be beneficial for companies working in the specific domains – a domain model is stable and commonly agreed, expert's knowledge is available. In case of entering new domain the company should work out specific domain model, which needs to be comprehensible and accepted by all developers. An evolvable domain model is a challenge which should be resolved in future works. Adding or removing some concepts from a conceptual domain model in the middle of project will raise the question what to do with the fragments which are already mapped to that concept. If a new concept is added the similarity between concept and closest fragments could be automatically calculated and the most related fragments re-mapped. Deletion should not remove the concept from domain model, but lock it not allowing mapping new fragments. This would preserve existing links between the concepts and fragments.

Change impact assessment is vital for the large development projects and perhaps the most risky and error-prone task. This approach enables to calculate the probability – how likely some product fragments will be impacted by the change of 'related' fragment. That value is calculated based on the weighted relations between domain concepts.

Huge domain model with thousands of concepts could be real challenge for developers to find relevant concept and to link a fragment in question. This issue can be solved by concepts clustering which could ease the finding the right concept. Development of currently hot research area in ontology mapping could also provide useful methods and techniques which could be used both to find the most relevant concept for the fragment and to develop stable and common agreed domain specific model when a domain is new for the developers and several interpretations of domain model exist.

6. Acknowledgment

Special thanks go to my colleague Sari Hakkarainen, who read and corrected the paper and helped to refine the ideas.

7. References

[1] T. Brasethvik and J. A. Gulla. "Natural Language Analysis for Semantic Document Modeling." In *Proceedings of the 5th International Conference on the Application of Natural Language for Information Systems* (NLDB'2000) in Versailles, France, June 2000

[2] A. Egyed, "Reasonings about Trace dependencies in a Multi-Dimensional Space", in *Proceedings of the 1st International Workshop on Traceability*, co-located with ASE 2002, Edinburgh, Scotland, UK, September 2002, pp. 42-45

[3] B.A. Farshchian, *A Framework for Supporting Shared Interaction in Distributed Product Development Projects*, PhD thesis, NTNU, Trondheim, Norway, 2001.

[4] S. T. Frezza, S. P. Levitan, P. K. Chrysanthis, "Linking requirements and design data for automated functional evaluation", Computers in Industry, Volume 30, Issue 1, Elsevier Science Publishers B. V., September 1996, pp. 13-25.

[5] O.C.Z. Gotel, A.C.W. Finkelstein, "An Analysis of the Requirements Traceability Problem", In *Proceeding of the 1st International Conference on Requirements Engineering* (ICRE'94), IEEE Computer Society Press, Colorado Springs, Colorado, USA, April 1994, pp. 94–102.

[6] S. Greenspan, C. McGowan, Structuring Software Development for Reliability, In *Microelectronics and Reliability*, 17, 1978 - p. 75–84.

[7] P. Grünbacher, A. Egyed, and N. Medvidovic, “Reconciling Software Requirements and Architectures - The CBSP Approach”, In *Proceedings of the 5th IEEE International Symposium on Requirements Engineering* (RE’01), Springer-Verlag, Toronto, Canada, 2001, pp. 202-211.

[8] F.V. Jensen, *An Introduction to Bayesian Networks*. UCL Press, London. 1996.

[9] P. Letelier, “A framework for Requirements Traceability in UML based projects”, in Proceedings of the 1st International Workshop on Traceability, co-located with ASE 2002, Edinburgh, Scotland, UK, September 2002, pp. 32-41.

[10] J. Pearl, *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*, Morgan Kaufmann. 1988.

[11] F. Pinheiro and J. Goguen. "An Object-Oriented Tool for Tracing Requirements". *IEEE Software*, 1996 13(2), pp. 52-64.

[12] K. Pohl, “PRO-ART: Enabling Requirements Pre-Traceability”, In *Proceedings of the Second International Conference on Requirements Engineering* (ICSE '96), Colorado, USA, 1996, pp. 76-85.

[13] K. Pohl, M. Brandenburg, A. Gülich, “Integrating Requirement and Architecture Information: A Scenario and Meta-Model Based Approach”, In *Proceedings of the Seventh International Workshop on Requirements Engineering: Foundation for Software Quality* (REFSQ'01), Interlaken, Switzerland, 2001.

[14] B. Ramesh and M. Jarke. "Toward Reference Models for Requirements Traceability". *IEEE Transactions on Software Engineering*, Vol. 27, No. 1, pp.58-93, January 2001.

[15] Rational Suite AnalystStudio. URL: http://www.rational.com/products/astudio/index.jsp

[16] D. Strašunskas, “Traceability in a Collaborative Systems Development from Lifecycle Perspective”, in *Proceedings of the 1st International Workshop on Traceability*, co-located with ASE 2002, Edinburgh, Scotland, UK, September 2002, pp. 54-60

[17] A. Sølvberg and T. Brasethvik, “The Referent Model Language”, Technical Report. NTNU, Trondheim, Norway URL: http://www.idi.ntnu.no/~ppp/referent/

[18] R. Watkins, M. Neal, “Why and How of Requirements Tracing”, *IEEE Software,* 1994 11(4), pp. 104-106.

F.3 Process of Product Fragments Management in Distributed Development

Strasunskas, D., and Hakkarainen, S. Process of Product Fragments Management in Distributed Development. In Meersman, R., Tari, Z., Schmidt, D. et al. (Eds.) *Proc. of the 11th Intl. Conf. on Cooperative Information Systems* (CoopIS'2003), Springer-Verlag, LNCS 2888, Catania, Sicily, Italy, November 2003, pages 218-234.

Process of Product Fragments Management in Distributed Development

Darijus Strašunskas and Sari Hakkarainen

Dept. of Computer and Information Science, Norwegian Univ. of Science and Technology
Sem Sælands vei 7-9, NO-7491 Trondheim, Norway
{dstrasun; sari}@idi.ntnu.no

Abstract. Management of product constituent fragments is essential for large scale logically or physically distributed projects. Geographically distributed development projects have special settings and needs – special attention has to be given to artifacts management because developers are likely to use different representation formats and a variety of tools for the artifact production. The question is: how can artifacts in different representation formats be related and managed?

Methodological support for artifacts management and traceability is presented in this paper. Product fragments from different development phases (i.e., requirements specification, design, code, test scenarios, and documentation) are interrelated through a conceptual domain model. Domain model is proposed as a means to capture information content despite heterogeneous representation. Given, a domain model with intra-related concepts and artifacts associated to the concepts we are able to interrelate heterogeneous artifacts and to predict and assess how one altered artifact may impact other artifacts. The approach covers the whole lifecycle of a system, enables artifacts' management by associating them according semantics contained inside.

1 Introduction

Information system development is a highly iterative process, in which developers seek to capture the needs and desires of all stakeholders. The goal is to transform the requirements into a complete system, consisting of both manual and computerized parts. The product of a development project undergoes changes because of its iterative nature. Management of the development process imposes requires fine-grained control over all fragments produced throughout whole lifecycle. Traceability facilitates product and process management and control. Traceability is [9] a property of a system description technique that allows changes in one of the system descriptions – requirements specification, design, code, documentation, or test scenarios – to be traced to the corresponding fragments of the other descriptions. Further, such correspondence relationships should be maintained throughout the life time of a system in order to manage the artifact.

Traceability has received attention in the requirements engineering literature [8] [21], where change management requires special efforts because of the highly iterative process and frequent re-conceptualizations. Especially, the pre-requirements traceability has been studied thoroughly [13] [14]. However, there is a lack of traceability tools to support the full life-cycle, starting from artifact inception through formalization process to its use. Different representation formats that are used throughout the development process make it complicated to cover the whole life-cycle of an artifact. Given, a single requirement maps to multiple architectural and design concerns which is used to derive, it is difficult to maintain the consistency and traceability. Moreover, an architectural or design component has a number of other relations to various requirements. The task becomes even more difficult in the face of a large system that is build to satisfy thousands of requirements.

System specifications consist of a wide variety of fragments (artifacts), i.e. different kinds of information about system that together comprise a full (or partial) system specification at various levels of abstraction. Some of these artifacts are well structured, textual or graphical documents, while others are more loosely structured. In a geographically distributed project developers may use different tools to create and modify product fragments. The fragments can be refined iteratively and further processed by colleagues. Afterwards produced fragments are interchanged among members of a project, so that is important for colleagues to interpret an artifact correctly. The main challenges are to interrelate and manage all artifacts in different representation formats that are produced in a distributed manner using different tools, and to cover the whole product lifecycle.

The objective of this work is to present an approach to product fragments management during the distributed collaborative development process. The assumptions are that there are intra-related concepts in the problem domain and fragments are mapped to them. Given those conditions, the semantics of an artifact are increased by the artifact mapping to the corresponding concepts, and that enables predicting and assessing fragment change impact on other fragments.

The paper is structured as follows. In section two, related work is analyzed. In section three, the domain model based approach for product fragments management is presented. In section four, a case study is applied and illustrated by using weighted graphs. Finally, in section five, the work is concluded and its possible shortcomings with some insight to how to solve them are discussed.

2 Related Works

Over the recent years, a number of techniques have been proposed to facilitate management of product development through traceability enabling techniques. Some examples are [8] cross referencing schemes, based on some form of tagging, numbering, or indexing; and requirements traceability matrices. Studies in the field of traceability have mainly focused on specific parts of the development process [18] – mostly in the areas of pre-requirements traceability (e.g. [13] [14]) and linking requirements to architectural components (e.g. [10], [15]).

Some of the approaches are based on a specific modeling language and /or a tool. A much cited tool is TOOR (Traceability of Object-Oriented Requirements) [13], which is based on FOOPS, a formal object-oriented language. Integrating textual specifications and UML (Unified Modeling Language) model elements is used by Letelier [11], as a framework for configuring requirements traceability. Both approaches are restricted to FOOPS and UML respectively and can sequentially only be applied to software process based on the same language.

Some approaches establish traceability links after the most of a system is developed (e.g. after producing requirements specification, code, etc.) and, per se, contribute mainly for product maintenance. Frezza et al [7] base their approach on simulation where both the requirements and the implemented system are simulated in order to obtain a set of result data. The data from the requirements and the implementation phase are then compared, which results in a quantitative measure of how accurate the running system implements the requirements. Egyed [5] uses a scenario driven approach to acquire runtime information about a system and relates the information – the footprints - to the requirements and a model of the running system. The footprints are analyzed in a tool, which shows how the components of the system interact when performing specified scenarios. Thus, provides additional trace information on how the running system actually fulfills its requirements and which parts of the design are affected.

Ramesh and Jarke in [16] offer a wide vision about the information that is needed for requirements traceability. Their study is based on an analysis of industrial software development projects. Two segments of traceability users are identified and two corresponding traceability meta-models are suggested. Proposed meta-models are extensive, but nevertheless do not show how different parts of system specification in various representation formats and abstraction levels can be related and traced. For instance, their rationale submodel includes decisions, issues or conflicts, assumptions, alternatives and arguments. This enables very precise description of the change necessity and situation at a particular time. However, recording of rationale has not been widely accepted in the industry due to the disruptive nature of recording the actions as they occur [1].

Hence, in overall, there is a lack of support to the whole product lifecycle. There is also an apparent lack of support for distributed teams that use different tools, representation techniques and notations. There exist development environments (e.g. Rational Suite AnalystStudio [17]), which compound together programs for requirements engineering, design, change management and code repository. Such environments are integrated programs that *a)* do not support collaborative work, *b)* do not support all project phases equally well, *c)* and a customer is bound to one vendor and language (environment) by choosing this kind of tool environment. Below, an attempt to fill in these gaps is presented.

3 Proposed Approach – Mapping to the Domain Concept

In this section we discuss our methodological approach for the artifacts management and traceability. First, the settings of the proposed approach are discussed. Next, functional perspective describes main steps required to enable and apply our approach. Finally, a meta-model describing the scope of the approach is presented and discussed.

3.1 Settings for the Approach

Above, it was argued that it is essential to enable change management and impact prediction through all phases of development in the distributed projects as mentioned above. To cover the whole lifecycle means that different tools and, most likely, different notations are used during the development project. A list of requirements for product development environments in order to enable collaboration in geographically distributed software products development is used in [6]. Here we adopt the requirements as follows.

Requirement 1. Unrestricted product object types – a product development environment should allow the developers to share any type of object that they might find useful for supporting their cooperation.

Requirement 2. Unrestricted relation types – a product development environment should allow the developers to create any type of relation between any two objects of product.

Requirement 3. Incremental product refinement – a product development environment should provide the developers with flexible mechanisms for incrementally refining the product. Hence, the developers should be allowed to start with vague products, and to refine them into more complete and formal ones.

The above three requirements were selected as to cover support for collaboration in distributed projects. Here, the product management and traceability method should meet the requirements 1 – 3 to ensure the applicability of the approach. As this approach is based on the fragments mapping to domain concepts, we say that a `fragment` is a well-defined piece of specification and has semantics, machine readable representation and identity, and supplementary, a `concept` is a well-defined unit of terms found in specific domain description. Further, there are two basic assumptions underlying the approach as follows.

Assumption 1. CASE-tools (Computer Aided Software Engineering) that are used during the product development support XML (eXtensible Markup Language) or XML-dialect format output of developed fragments.

Assumption 2. There is a problem domain and it can be characterized by well-defined, interrelated concepts. Furthermore these concepts are represented as nodes having weighted relationships which show the strength of relationship between the concepts (relatedness of concepts).

The former assumption is reasonable, since most CASE-tools maintain model interchange formats derived from XML and the latter is more restrictive since not all relationships can be easily expressed by weights.

3.2 Functional Perspective of the Approach

Based on above described assumptions, the overall process (see fig.1) consists of four basic steps, where the last three steps are iterative.

Step 1 – Building of conceptual domain specific model. This step consists of two main sub-steps: *(a)* extraction of domain specific concept and *(b)* weighing of relationships between concepts.

Step 1.a – Syntactical analysis of textual documents has been investigated thoroughly in last few decades. Natural language processing is a main technique used to extract more structural information out of documents. Efforts are directed to build models from requirements specification in natural language. The naïve approach is to use nouns as candidates for entities and verbs for relations between entities. However, there is necessity for more sophisticated techniques to handle linguistic variation when proposing model elements when constructing domain models from a large set of documents. [2] proposes approach of natural language analysis for semantic documents modeling, where techniques for domain model construction are discussed. The natural language based approach is adapted for concept extraction.

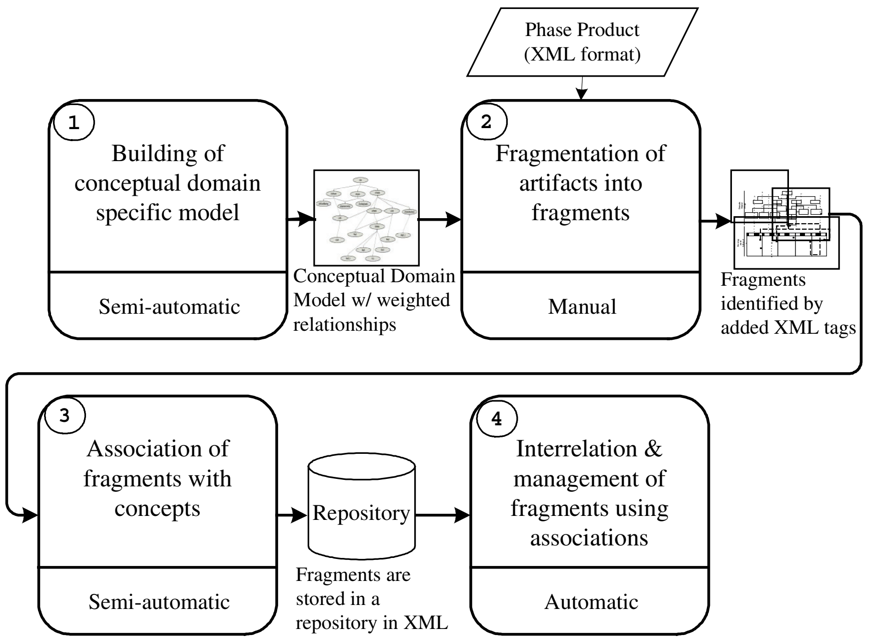

Phase Product (XML format)
1
Building of conceptual domain specific model
Semi-automatic
Conceptual Domain Model w/ weighted relationships
2
Fragmentation of artifacts into fragments
Manual
Fragments identified by added XML tags
3
Association of fragments with concepts
Semi-automatic
Repository
Fragments are stored in a repository in XML
4
Interrelation & management of fragments using associations
Automatic

Fig. 1. Main steps to enable fragment management

Step 1.b – Quantification of the relationship between concepts is supported by using linguistics and natural language processing techniques for analyzing the documents from a domain. Collocation technique and text mining are used to evaluate the strength of relationship between concepts. The values should be refined by the domain expert – this reflects domain expert's belief in how much concepts are related in a particular domain. So, these numbers come from either objective data or the experiences of the domain expert accumulated from the development of similar projects. These ranges can be used to represent the high[1] (0.0 to 0.3), medium (0.4 to 0.6) and low (0.7 to 1.0) relatedness degree.

Step 2 – Fragmentation of artifacts into fragments. Produced artifact is translated to XML format and logically fragmented according its semantics. Fragmentation is done by a traceability module which gets the XML file as input and provides means for developer to define boundaries of a fragment. An XML file with identifying tags for start and end positions of fragment is produced as output.

Step 3 – Association of fragments with the concepts. Candidate concepts form domain model are suggested automatically by processing the fragments. Techniques

[1] 'High' means that distance between concept and fragment is short. The values are application sensitive, see a case study in chapter 4.

from *Step 1* are adapted to extract concepts, if possible, from the fragments and propose the closest related concept from domain model to map to it. Fragments can be linked directly to other fragments if developer finds them related or one fragment is part of another (more detailed explanation is provided in the meta-model description below). The weighing scheme is used as described in *Step 1.b.* The mapping rate is revised and confirmed by the developer, who created new or a version of the fragment and checked-in to the repository. The relationship information is encoded using XML tags. Finally, the fragments are stored in a central repository.

Fig. 2 presents a part of RML (Referent Model Language) [20] model in XML format where boundaries of a semantic fragment are identified by the tags `<fragment id="R0012">` and `</fragment>`, and the semantics of the fragments is encoded within the tags `<semantic-association>` and `</semantic-association>` by the associated concepts `<concept id="c17", weight="0.7"/>` and `<concept id="c05", weight="0.9"/>`.

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<referent-diagram>
  <generator app="refedit" version="Version: 2.3c" ver
   date="Sun 6 Feb 2000">
          <fragment id="R0012">
            <semantic-association>
              <concept id="c017" weight="0.7"/>
              <concept id="c005" weight="0.9"/>
            </semantic-association>
                <content>
                  <referent id="x1">
                       <position x="626" y="83" />
                       <dimension width="109" height="41" />
                    <text>
                       <position x="667" y="98" />
                       <string>Product</string>
                    </text>
                    <aggregation id="x2" idref="x1">
                       <position x="667" y="169" />
                    </aggregation>
                  </referent>
                </content>
          </fragment>
...
</referent-diagram>
```

Fig. 2. Cutout of fragmented RML model in XML representation

Step 4 – Usage of the associations for fragments interrelation and management. Domain model is constructed and concepts in the model are intra-related by weighted links according how strongly concepts relate. Those weights are further used to evaluate interrelations between fragments mapped to the domain concepts and to estimate likelihood of impact of one fragment to another.

Thus, association relations are based on the semantics of the artifacts. Fragments are linked to the concepts from the domain model; all selected fragments are mapped and linked through the conceptual domain model as follows.

There exists a set of concepts {C_1, C_2, …, C_n} and a set of fragments {F_1, F_2, …, F_m}, then consequently:

– If fragment F_i is mapped to a concept C_i and fragment F_j is mapped to C_i, then transitively F_i also relates to F_j:

$$(\mathbf{F_i} \rightarrow \mathbf{C_i}) \wedge (\mathbf{F_j} \rightarrow \mathbf{C_i}) \Rightarrow \mathbf{F_i} \rightarrow \mathbf{F_j} . \quad (1)$$

– Given, the related concepts C_i and C_j, and if fragment F_i is linked to a concept C_i and a fragment F_j is linked to C_j, then trace dependency to certain degree exists between F_i and F_j.

$$(\mathbf{C_i} \rightarrow \mathbf{C_j}) \wedge (\mathbf{F_i} \rightarrow \mathbf{C_i}) \wedge (\mathbf{F_j} \rightarrow \mathbf{C_j}) \Rightarrow \mathbf{F_i} \rightarrow \mathbf{F_j} . \quad (2)$$

3.3 Meta-model of the Approach

The scope of the approach based on the above settings is specified in a meta-model using RML [19, 20] (see fig. 3). We deal with product development, using system development tools (Syst.Dev.Tool), where a system development tool can also be seen as product, when it is under development. Every product development has a specific lifecycle consisting of different phase type (e.g., business analysis, requirements engineering, design, implementation, testing, etc.). Each phase type has a distinct phase product (e.g. requirements specification, design, code, user manual, and software itself), which is result of particular lifecycle phase. A product is final result of the development project, and it consists of the interrelated phase product.

A fragment is a semantic piece of phase product in a certain level of granularity, e.g., it can be a document, a model, a diagram, a section in a document, a text specifying a non-functional requirement, an use case, a class, an attribute, etc. Fragment can be composed of fragments. Such a fragment is inreflexive, asymmetric, and non-transitive. Fragment can have a direct dependence link to another fragment. Every fragment has semantics, which relate the fragment to one or more concept. A rated mapping relationship is used to distinguish fragment coherency to a particular concept. Semantics of certain fragments can be best described by several concepts or a particular concept cluster, which groups related concepts and composes the domain model. Concepts are connected by an undirected graph with weights (weighted relationship). Weights of those relations are calculated based on the degree of the concept relatedness.

Since recording of rationale is not widely accepted in the industry due to the disruptive nature of recording the actions as they occur [1], the attempt for extensive trace information record can be crucial in huge distributed development projects. Therefore, we expect only vital trace info to be captured. We keep track on the evolution of fragment, the direct relationship between fragments and the relationship between fragment and concept cluster by recording the following information – rationale, change operation (i.e., addition, deletion, altera-

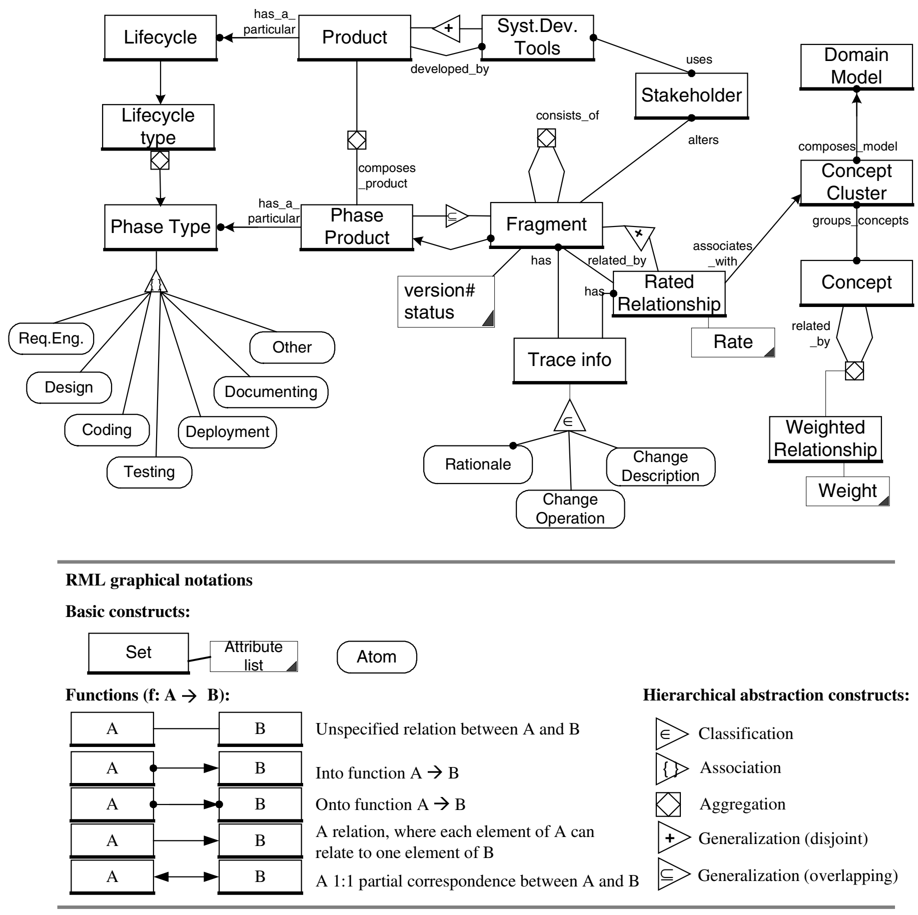


Fig. 3. Meta-model of fragment interrelation and management

tion), and `change description`. Additionally fragment has `version number` and `status`, this information is used for compositional fragment and phase product configuration management. Three configurations are important [18], namely 1) the *latest project-wide*; 2) *latest consistent*; 3) *latest authorized* configuration. The latest project-wide baseline reflects all new developments in the project. Latest baselines are not supposed to be consistent, but all new ideas since the last logon are detected by inspecting this configuration. Latest consistent baselines represent the most recent stable work. These configurations are considered to be candidates for authorization. Latest authorized is the last agreed configuration and it forms the official basis for all subsequent work of the development group.

Fragments interrelationship and trace information are added as metadata (information/data about data) to abstract away from the heterogeneous representation details and capture information content. Association of each product fragment with the corresponding concepts from a domain model enriches its semantic meaning. The intuition is that the more explicit information a fragment conveys, the better the chance that it will be interpreted correctly by other stakeholders. More precise relationship is captured by direct linking between related fragments. Since that is not trivial task even in a smaller scope projects, we see it important to have them at some certain stage of the project. Establishment of direct linking is done in few steps. The initial one is, of course, fragment association with domain concept. Next, by exploitation of those relationships is means for change impact prediction and assessment (see a case study section). When developer alters the fragment because of the change made in another fragment, then establishment of the direct linking between those two fragments is suggested automatically. In this way, we are able incrementally refine and establish fine-grained traceability information between fragments.

4 Application of the Approach

In this section we present a case example to test practical applicability and illustrate the proposed approach in empirical settings. This is done for better presentation and communication of the idea. Description of application of the approach consists of the case study and candidate technique – weighted graphs.

4.1 Weighted Graphs

Weighted graphs are used to represent a concept model. Interrelation of the concepts is depicted as a semantic distance between concepts. The shortest path algorithm is used to predict which fragments are most likely to be impacted.

Given, `G` is a weighted graph. The length (or weight) of a path `P` is the sum of the weights of the edges of `P`. That is, if `P` consists of edges e_0, e_1, …, e_{k-1} then the length of `P`, denoted `w(P)`, is defined as

$$w(P) = \sum_{i=0}^{k-1} w(e_i) \ . \tag{3}$$

The distance from a node `v` to a node `u` in `G`, denoted `d(v, u)`, is the length of a minimum length path from `v` to `u`, if such path exists. We calculate a shortest path (i.e., using algorithms for single-source shortest path, for instance, Dijkstra algorithm [4] or Bellman-Ford [3]) from some node `v` (usually, that is the fragment, which has been altered) to each other node in `G`, viewing the weights on the edges as distances.

4.2 A Case Study

Domain description. A case study is based on MEIS system, used for the basic course of information systems SIF8035 [12]. MEIS system is used for exercise delivery and evaluation. There exist two groups of users: students (they are also reviewers of others' solutions) and student assistants, who check all deliveries (both solutions to exercise and evaluation of those solutions) and either accept or reject them. Main domain concepts and relationships among them are depicted in the fig.4. Quantification of relationships between concepts (semantic distance) has been performed manually relying on the knowledge of domain. Weights used are from the range `[0.0, 1.0]`, where `0.0` means that concepts have high semantic relatedness in the domain, and the value `1.0` means, that the semantic distance between concepts is very long (concepts are not related at all). For instance, the weight of relationship between '`Student`' and '`Reviewer`' is equal to `0.0` only in this domain, where students are also reviewers of others' solutions.

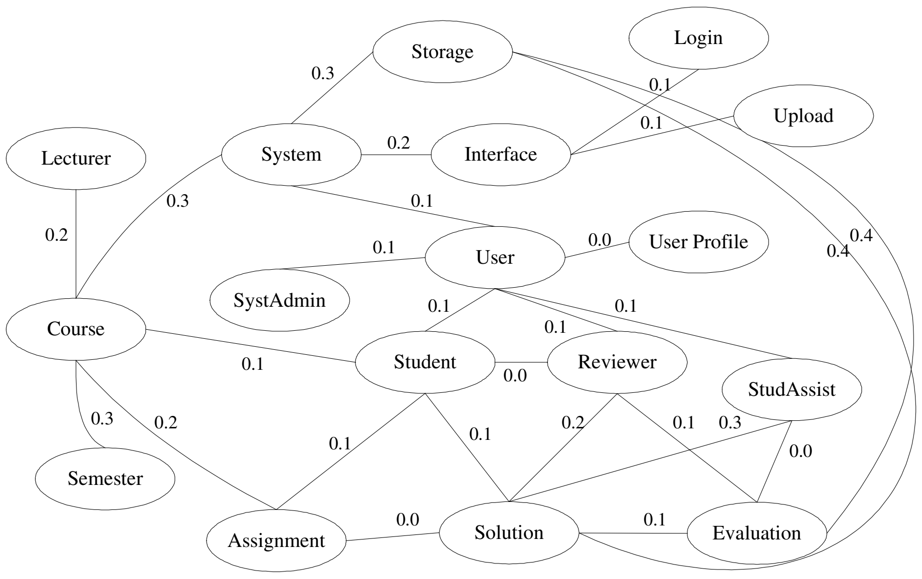


Fig. 4. Domain model for MEIS

Fragmentation. During the development of the MEIS system every requirement was treated as a separate fragment. Some of them are listed below and other kinds of product fragments (use case, code, design, user interface) are presented in figures 5-8. Requirements for MEIS system:

Req.1. It should be possible to create users' profiles from textual file.

Req.2. Student should be able to upload solution:

Req.2.1. Solution should be stored in the student's folder.

Req.2.2. Reference (link) to solution1&2 should be kept in the MEIS database.

Req.3. StudAssist should accept/reject a solution1&2.

Req.3.1. System should provide possibility to reject solution1&2.

Req.4. StudAssist should form a reviewer groups for solution1&2.

Req.4.1. System should provide to StudAssist a list of students, whose solution was accepted.

Req.4.2. StudAssist should form a reviewer group.

Req.5. Reviewer should deliver evaluations of solution1 and solution2.

Req.5.1. `Reviewer should evaluate DFD/APM model of solution1&2.`

Req.5.2. `Reviewer should upload Word documents with evaluation for DFD/APM model of solution1&2.`

Req.5.3. `File with evaluation for DFD/APM model of solution1&2 should be stored in the database.`

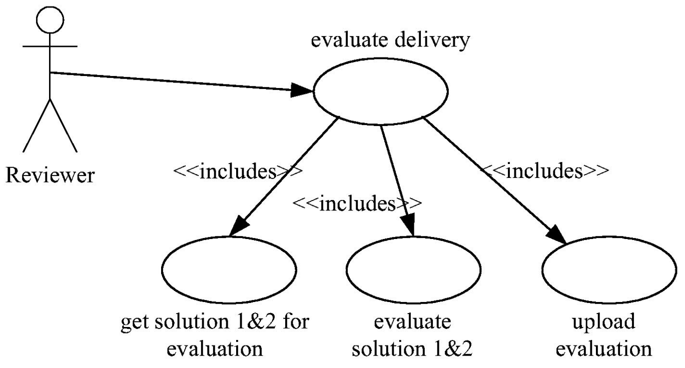
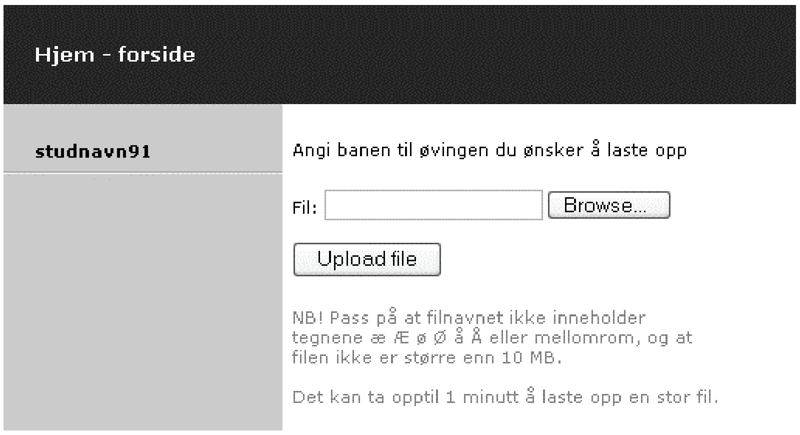


Fig. 5. Fragment – Use Case diagram – reviewer tasks ('UC.1' in fig.9)

```
if ((dbproc = mysql_init(NULL)) == NULL) {
    printf("Unable to init.\n<hr>");
  } else {
    if (mysql_real_connect(dbproc, NULL, "ADMIN_USER",
                           "ADMIN_PASSWORD", "DATABASE_NAME"
                           0, "MYSQL_SOCK", 0) == NULL) {
      printf("Unable to connect.\n<hr>");
    } else {
      if (is_modify) {
        passwd = Find_Value(entries, num_words, "Password1")
        if (passwd == NULL || strlen(passwd) <= 0) {
          sprintf(passwd_buf, "'%s'",
                  Find_Value(entries, num_words, "Password")
        } else {
          sprintf(passwd_buf, "PASSWORD('%s')", passwd);
        }
```

Fig. 6. Fragment – part of code ('Code.1' in fig.9)

Association with a concept. As described in the previous section, developers use the tool for semi-automatic fragments mapping to domain concepts. Additional XML tags are added to keep information about the related concepts and weight of relationship, as a fragment could have one or more related concepts (recall fig. 3). For example, requirement '`Req.5.2: Reviewer should upload Word documents with evaluation for DFD/APM model of solution1/2`' provides hints about relation to the concepts '`Reviewer`', '`Upload`', '`Evaluation`' and '`Solution`'. Never-



Hjem - forside

studnavn91

Angi banen til øvingen du ønsker å laste opp

Fil: Browse...

Upload file

NB! Pass på at filnavnet ikke inneholder tegnene æ Æ ø Ø å Å eller mellomrom, og at filen ikke er større enn 10 MB.

Det kan ta opptil 1 minutt å laste opp en stor fil.

Fig. 7. Interface screenshot (‘Doc.1’ in fig.9)

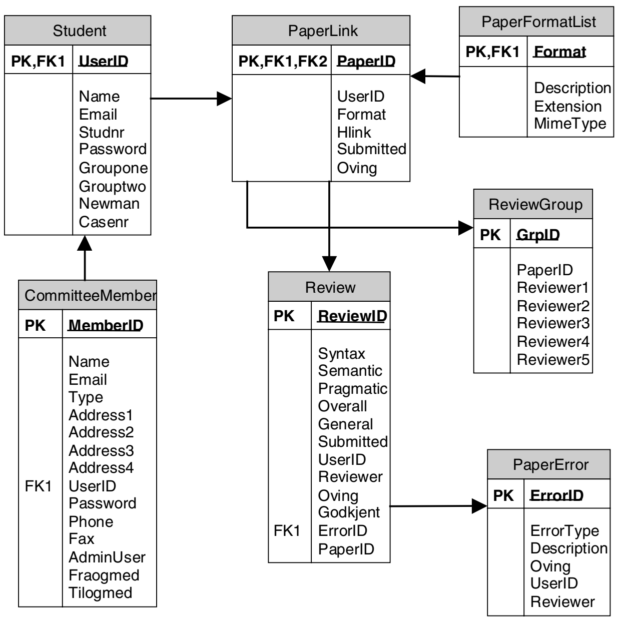


Fig. 8. Fragment – ER diagram of MEIS database (‘Dsgn.1’ in fig.9)

theless, it is mainly about ‘evaluation upload’, so this requirement is mapped to the concepts ‘`Upload`’ and ‘`Evaluation`’ with the assigned weights[2] `0.1` and `0.3` re-

[2] Fragment association to concept and weight assignment is more intuition based. Developer knows best the semantics of the fragment. To facilitate the task for developer in assigning the value, only three values are used to identify the relatedness of the concepts – high, medium and low (recall *Step 1.b*)

spectively. Partial[3] graphical representation of the fragments mapped to domain model is depicted in fig.9. The concepts and fragment from above described example are gray shaded. It should be noted that fig. 9 does not imply the way for fragments mapping, but is used here only for explanatory purposes.

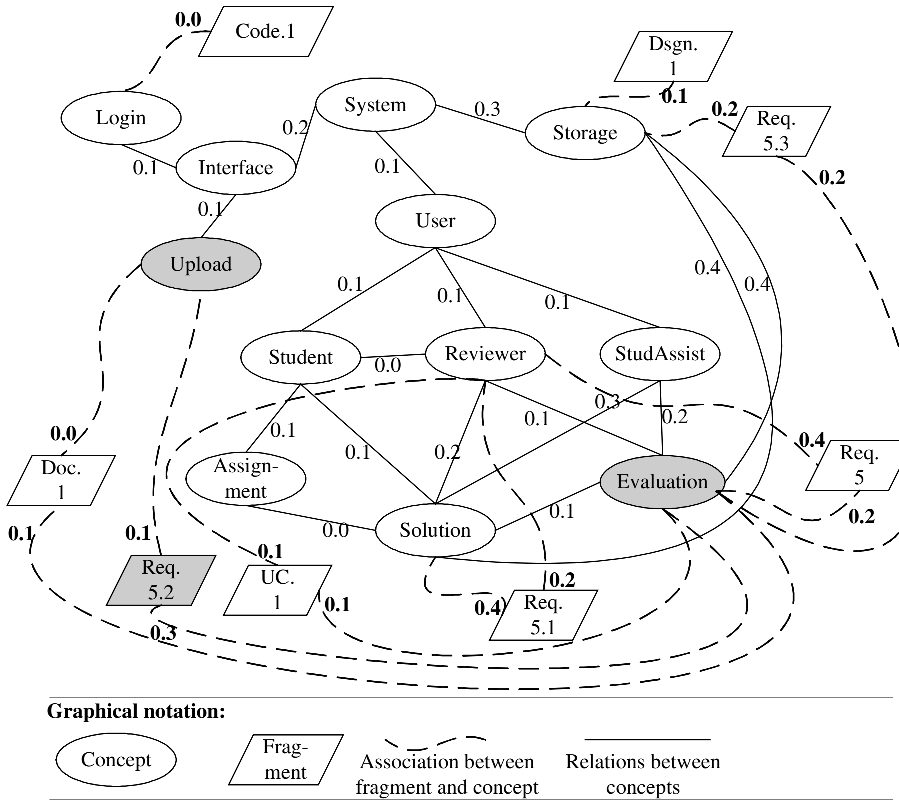


Fig. 9. Graphical representation of MEIS fragments mapping to domain concepts (partial)

For better explanation, table 1 shows mapping the fragments to domain concepts and distance (weight) between fragment and particular concept.

Alteration. During the system development it was decided to make standard web form for evaluation instead of delivering evaluation in Word file. As consequence requirement (`Req.5`) has been changed to:

```
Req.5. Reviewer should be provided 2 (two) web forms for
evaluation of each solution1/2.
```

[3] Concepts, which are not associated to any fragment, are removed from fig.9 (in comparison with fig.4) with a reason not to introduce cognitive overload on the reader. As well as partial association of only few fragments is shown.

Table 1. Association and relatedness of the fragments to the concepts

Fragment	Relevant Concepts					
	Storage	Login	Upload	Reviewer	Solution	Evaluation
Req.5				0.4		0.2
Req.5.1				0.2	0.4	
Req.5.2			0.1			0.3
Req.5.3	0.2					0.2
UC.1				0.1		0.1
Dsgn.1	0.1					
Code.1		0.0				
Doc.1			0.0			0.1

This alteration is recorded and saved into the repository. The vital information, which needs to be captured, was discussed in the meta-model description. Fig.10 illustrates this captured trace information, i.e. who altered 'source' fragment, what change operation was performed, what was done and what was rationale behind that change.

```
<fragment id="R005" version="v.1.2" status="authorized">
  <change operation="alteration">
    <user id="dstrasun">
    <rationale> It is necessary to change delivery way in or-
                der to enable automatic comparison of different
                evaluations</rationale>
     <description> Requirement to upload the evaluations in a
                word file was changed to provide web form for
                the evaluation</description>
  </change>
</fragment>
```

Fig. 10. Trace information about the change in XML representation

Assessment of the impact probabilities on other fragments caused by this change is shown in Table 2. Results are calculated applying Eq.3 and using weights between fragments and concepts, and weights between concepts, as a distance between points (nodes). For example, the distance between fragments 'Req.5' and 'UC.1' was calculated in the following way: 'Req.5' is associated with concept 'Evaluation' with a value of 0.2, and 'UC.1' is associated with the same concept with a value of 0.1, so the path (semantic distance) from the altered fragment 'Req.5' to probably impacted fragment 'UC.1' is equal to 0.3.[4]

Since altered fragment is associated with 2 concepts, namely 'Evaluation' and 'Reviewer'. Shortest paths are computed going through both concepts. The purpose for that is to reduce impact probability warnings by allowing developer to specify what part of fragment's semantics was changed. For example, it is obvious that requirement change does not effect reviewer (as he/she still should deliver evaluation),

[4] Fragments associated with the same concept, usually will have the shortest path, or semantic distance, as this mapping to the same concept shows that semantics of those fragments are almost the same.

but only the form and way of evaluation. Allowing to developer specify that, we decrease the impact warnings – that means, that only inference through the concept '`Evaluation`' should be taken into account and checked (see 2nd column of table 2). It means that developers should go through and check for consistency the top-ranked fragments in 2nd column. If impacted semantics are not specified, then the weighted average can be used.

As all mapped fragments are being assessed, only the ones with the shortest path, i.e. when fragments are very close semantically, should be checked for impact. Of course, there should be defined threshold for notification posting in large development project, threshold value depends on specific project settings and requires attentive empirical study. Defining the threshold to 0.5, these 4 fragments need to be checked for consistency with the change performed: '`UC.1`', '`Doc.1`', '`Req.5.3`', and '`Req.5.2`'.

Table 2. Impact assessment based on calculation of shortest path

Fragment	Concepts		
	Evaluation	Reviewer	Average
UC.1	0.3	0.5	0.4
Doc.1	0.3	0.9	0.7
Req.5.3	0.4	1.1	0.9
Req.5.2	0.5	1.0	0.8
Req.5.1	0.7	0.6	0.6
Dsgn.1	0.7	1.0	0.9
Code.1	0.8	0.9	0.9

5 Concluding Remarks and Future Work

In this paper we have described the methodological approach to enable product fragment management in the distributed system development projects. Proposal is based on semantics enrichment of the produced fragments by mapping them to related concepts from specific domain model. These inter-relations are weighted as well as intra-relations among the concepts in a domain model. Weights assigned to relationships suit as basis for impact prediction and assessment.

The approach (a) enables whole lifecycle product management. As nature of collaborative development is usually very iterative, the approach (b) allows relating product fragments at different stages of its incremental refinement (e.g., from abstract sketches to formal representation), (c) does not bind developers to a specific tool and/or modeling language, as far as used tool supports XML output. The use of XML makes it possible to use this approach in settings where the involved artifacts are created and managed by heterogeneous tools, such as text processors and CASE-tools.

Proposal can be beneficial for companies working in the specific domains – a domain model is stable and commonly agreed, expert's knowledge is available. In case of entering new domain the company should work out domain model, which needs to be comprehensible and accepted by all developers. An evolvable domain model is a challenge which should be resolved in future works. Adding or removing some concepts from a conceptual domain model in the middle of project will raise the question what to do with the fragments which are already mapped to that concept. If a new concept is added the relatedness between concept and closest fragments could be automatically calculated and the most related fragments re-mapped. Deletion should not remove the concept from domain model, but lock it not allowing to associate new fragments. This would preserve existing links between the concepts and fragments.

Further, large domain model with thousands of concepts could be real challenge for developers to find relevant concept and to link a fragment in question. This issue can be solved by concepts clustering which could ease the finding the right concept. Development in the area of ontology mapping could also provide useful methods and techniques which could be used both to find the most relevant concept for the fragment and to develop stable and common agreed domain specific model when a domain is new for the developers and several interpretations of domain model exist.

However, the most important contribution of this paper is management of heterogeneous product fragments by interrelating them according their semantics and usage of those interrelations for change impact assessment. Change management and assessment is vital for the large development projects and perhaps the most risky and error-prone task. This approach enables to calculate the probabilities as semantic distance between heterogeneous product fragments – how likely some product fragments will be impacted by the change of ‘related’ fragment. That value is calculated based on the weighted relations between domain concepts and those weights depends on experts’ knowledge of the domain. As the calculation based on those weights is a backbone of this approach, the process of weight assignment should be well reasoned and methodologically described – big challenges for future works lie here.

Direct linking between related fragments would result in more precise relationship and change impact assessment. That is not trivial task even in a smaller scope projects and, certainly, more challenging in distributed development. Thus, we see it being important to refine the mechanism of direct links establishment between related fragments based on change impact history, i.e. when developer alters the fragment because of the change made in another fragment, then establishment of the direct linking between those two fragments should be suggested automatically.

References

1. Arkley, P., Mason, P., Riddle, S.: “Enabling Traceability”, in Proceedings of the 1st International Workshop on Traceability, co-located with ASE 2002, Edinburgh, Scotland, UK, September (2002) pp. 61–65
2. Brasethvik, T. and Gulla, J.A.: "Natural Language Analysis for Semantic Document Modeling." In Proceedings of the 5th International Conference on the Application of Natural Language for Information System*s* (NLDB'2000) in Versailles, France, June (2000)
3. Cormen, T.H., Leiserson, C.E., Rivest, R.L. and Stein, C.: Introduction to Algorithms, 2nd Edition. The MIT Press and McGraw-Hill, (2001)
4. Dijkstra, E.W.: A note on two problems in connextion with graphs. Numer. Math. 1:269–271, (1959)
5. Egyed, A.: “Reasonings about Trace dependencies in a Multi-Dimensional Space”, in Proceedings of the 1st International Workshop on Traceability, co-located with ASE 2002, Edinburgh, Scotland, UK, September (2002) pp. 42–45

6. Farshchian, B.A.: A Framework for Supporting Shared Interaction in Distributed Product Development Projects, PhD thesis, NTNU, Trondheim, Norway, (2001)
7. Frezza, S.T., Levitan, S.P., Chrysanthis, P.K.: “Linking requirements and design data for automated functional evaluation”, Computers in Industry, Volume 30, Issue 1, Elsevier Science Publishers B. V., September (1996) pp. 13–25
8. Gotel, O.C.Z., Finkelstein, A.C.W.: “An Analysis of the Requirements Traceability Problem”, In Proceeding of the 1st International Conference on Requirements Engineering (ICRE’94), IEEE Computer Society Press, Colorado Springs, Colorado, USA, April (1994) pp. 94–102
9. Greenspan, S., McGowan, C.: Structuring Software Development for Reliability, In Microelectronics and Reliability, 17, (1978) pp. 75–84
10. Grünbacher, P., Egyed, A. and Medvidovic, N.: “Reconciling Software Requirements and Architectures - The CBSP Approach”, In Proceedings of the 5th IEEE International Symposium on Requirements Engineering (RE’01), Springer-Verlag, Toronto, Canada, (2001) pp. 202–211
11. Letelier, P.: “A framework for Requirements Traceability in UML based projects”, in Proceedings of the 1st International Workshop on Traceability, co-located with ASE 2002, Edinburgh, Scotland, UK, September (2002) pp. 32–41
12. Matulevicius, R.: MEIS requirements specification. Technical report, NTNU, (2003)
13. Pinheiro, F. and Goguen, J.: "An Object-Oriented Tool for Tracing Requirements". *IEEE Software*, 13(2), (1996) pp. 52–64
14. Pohl, K.: “PRO-ART: Enabling Requirements Pre-Traceability”, In Proceedings of the Second International Conference on Requirements Engineering (ICSE '96), Colorado, USA, (1996) pp. 76–85
15. Pohl, K., Brandenburg, M., Gülich, A.: “Integrating Requirement and Architecture Information: A Scenario and Meta-Model Based Approach”, In Proceedings of the Seventh International Workshop on Requirements Engineering: Foundation for Software Quality (REFSQ'01), Interlaken, Switzerland, (2001)
16. Ramesh, B. and Jarke, M.: "Toward Reference Models for Requirements Traceability". IEEE Transactions on Software Engineering, Vol. 27, No. 1, pp.58–93, January (2001)
17. Rational Suite AnalystStudio. URL: http://www.rational.com/products/astudio/index.jsp
18. Strasunskas, D. “Traceability in a Collaborative Systems Development from Lifecycle Perspective”. In Proceedings of the 1st International Workshop on Traceability, co-located with ASE 2002, Edinburgh, Scotland, UK, September (2002) pp. 54–60
19. Solvberg A.: Data and what they refer to. In; Chen, P., Akoka, J., Kangassalo, H., Thalheim, B., (eds.): Conceptual Modeling: Current Issues and Future Trends. LNCS 1565. Springer Verlag (1999)
20. Solvberg, A. and Brasethvik, T.: “The Referent Model Language”, Technical Report. NTNU, Trondheim, Norway URL: http://www.idi.ntnu.no/~ppp/referent/
21. Watkins, R., Neal, M.: “Why and How of Requirements Tracing”, IEEE Software, 11(4), (1994) pp. 104–106

F.4 Domain Model Driven Approach to Change Impact Assessment

Strasunskas, D., and Hakkarainen, S. Domain Model Driven Approach to Change Impact Assessment. In Linger, H. et al. (Eds.), *Constructing the Infrastructure for the Knowledge Economy: Methods and Tools, Theory and Practice*[23], Kluwer Academic / Plenum Publishers, 2004, pages 305-316.

[23] Proceedings of the 12th International Conference on Information Systems Development (ISD'2003), Melbourne, Australia, August 2003.

DOMAIN MODEL DRIVEN APPROACH TO CHANGE IMPACT ASSESSMENT

Darijus Strašunskas and Sari Hakkarainen*

1. INTRODUCTION

Information system development is a highly iterative process in which developers seek to capture the needs and desires of all stakeholders. The goal is to transform the requirements into a complete system consisting of both manual and computerized parts. The product of such a development project undergoes changes because of its iterative nature. Extensive attention is given to traceability as a means to relate different different system descriptions and to allow changes in one of the system descriptions – requirements specification, design, code, documentation, or test scenarios – to be predicted and traced to the corresponding fragments of the other descriptions[1]. Such correspondence relationships should be maintained throughout the lifetime of a system in order to manage the artifact.

Change impact management and change propagation have received much attention in the requirements engineering literature[2, 3], as changes during requirements elicitation process are continual. However, there is a lack of tools to support the full lifecycle, starting from artifact inception to its use. Different representation formats that are used throughout the development process make it complicated to cover the whole lifecycle of an artifact. Given, that a single requirement maps to multiple architectural and design concerns, which are used to derive it, it is difficult to maintain the consistency and traceability between the fragments. Moreover, an architectural or a design component has a number of other relations to various requirements. The task becomes even more difficult in the face of a large system that is being build to satisfy thousands of requirements.

In a geographically distributed project, developers may use different tools to create and modify product fragments, which can be refined iteratively and processed further by colleagues. After the system descriptions are produced, they are interchanged and shared among members of the project, which places elaborate requirements on that colleagues interpret artifact correctly. The main challenges are to relate all artifacts in different representation formats that are produced in a distributed manner using different tools and to cover the whole product lifecycle.

* Dept. of Computer and Information Science, Norwegian Univ. of Science and Technology, NO-7491 Trondheim, Norway; dstrasun@idi.ntnu.no and sari@idi.ntnu.no

The objective of this work is to present an approach to product fragments[†] management and change impact assessment in distributed collaborative development process. The assumptions are that there are related concepts in the problem domain and that the fragments can be associated with them. Given that, the semantics of an artifact is increased by the artifact mapping to the corresponding concepts, which enables predicting and assessing fragment change impact on other fragments.

The overall structure of the reminder of this paper is as follows. In section two, related work is analyzed. In section three, the domain model driven approach to enable change impact assessment by relating fragments in different representation formats is presented. In section four, a case study is applied and illustrated by using Bayesian Belief Networks as a candidate technique for quantitative analysis. Finally, in section five, the work is concluded and its possible shortcomings with some insight on how to solve them are discussed.

2. RELATED WORK

Over the recent years, a number of techniques have been proposed for providing traceability and facilitating change management. Some examples[2] are cross referencing schemes, based on some form of tagging, numbering, or indexing and some are requirements traceability matrices. Studies in the field of traceability have mainly focused on specific parts of the development process[4] – mostly in the areas of pre-requirements traceability[5,6], and linking requirements to architectural components[7, 8].

Some of the approaches are based on a specific modeling language and /or a tool. A much cited tool is TOOR (Traceability of Object-Oriented Requirements)[5], which is based on FOOPS, a formal object-oriented language. Integrating textual specifications and UML (Unified Modeling Language) model elements is used by Letelier[9], as a framework for configuring requirements traceability. Both approaches are restricted to FOOPS and UML respectively and can subsequently only be applied to software processes based on the same language.

Some approaches establish links among dependent fragments after most of the system is developed, i.e., after producing requirements specification, code, etc., and, per se, contribute mainly for product maintenance. Frezza et al[10] base their approach on simulation where both the requirements and the implemented system are simulated in order to obtain a set of result data. The data from the requirements and the implementation phase are then compared, which results in a quantitative measure of how accurate the running system implements the requirements. Egyed[11] uses a scenario driven approach to acquire runtime information about a system and relates the information – the footprints - to the requirements and a model of the running system. The footprints are analyzed in a tool, which shows how the components of the system interact when performing specified scenarios. Thus, it provides additional trace information on how the running system actually fulfills its requirements and which parts of the design are affected.

In summary, existing approaches fall into two categories: (a) specific notation dependent and (b) post-analytical. The usage of the former enforces developers to learn a new language, which is expensive and error-prone. Usually, these approaches are created with a special purpose and cover only part of system development lifecycle. The latter

[†] We will use notions of "fragment" and "artifact" interchangeably in this paper.

group of the approaches contributes mainly to system maintenance. So, there is a lack of support for the whole product lifecycle. There is also an apparent lack of support for distributed teams that use different tools, representation techniques and notations. Below, an attempt to fill in these gaps is presented.

3. PROPOSED APPROACH - MAPPING TO THE DOMAIN CONCEPT

It is essential to enable change notification and impact prediction through all phases of development in the distributed projects as mentioned above. To cover the whole lifecycle means that different tools and, most likely, different notations are used during the development project. A list of requirements for product development environments in order to enable collaboration in geographically distributed software products development[12] is described by Farshchian. Here we adopt the requirements (**$\mathbf{Req_n}$**) as follows.

$\mathbf{Req_1}$ - Unrestricted product object types – a product development environment should allow the developers to share any type of objects that they might find useful for supporting their cooperation.

$\mathbf{Req_2}$ - Unrestricted relation types – a product development environment should allow the developers to create any type of relation between any two objects of product.

$\mathbf{Req_3}$ - Incremental product refinement – a product development environment should provide the developers with flexible mechanisms for incrementally refining the product. Hence, the developers should be allowed to start with vague products and to refine them into more complete and formal ones.

The above three requirements were selected in order to cover support for collaboration in distributed projects. Here, the method should meet the requirements **$\mathbf{Req_1}$** to **$\mathbf{Req_3}$**. As this approach is based on the fragments mapping to domain concepts, we say a `fragment` is a well-defined piece of specification and has semantics, machine readable representation and identity, and a `concept` is a well-defined unit of terms found in a specific domain description. Further, there are two basic assumptions (**$\mathbf{Assmp_n}$**) underlying the approach as follows.

$\mathbf{Assmp_1}$ - CASE-tools (Computer Aided Software Engineering) that are used during the product development support an XML (eXtensible Markup Language) or an XML-dialect format output for the developed fragments.

$\mathbf{Assmp_2}$ - There is a problem domain and it can be characterized by well-defined, interrelated concepts. Furthermore, these concepts are represented as nodes having weighted relationships, which show the strength of the relationship between the concepts, i.e., a relatedness value between the concepts.

The former assumption is reasonable since most CASE-tools maintain model interchange formats derived from XML and the latter is more restrictive since not all the relationships can easily be expressed by weights and the domain model should be shared and agreed by all participants. Based on these assumptions, the overall process (figure 1) applied in this approach consists of three basic steps (**$\mathbf{Step_n}$**):

$\mathbf{Step_1}$ - Building a conceptual domain specific model. This step consists of two main sub-steps: ***$\mathbf{Step_{1a}}$*** - extraction of domain specific concepts and ***$\mathbf{Step_{1b}}$*** - weighing of relationships between concepts.

$\mathbf{Step_{1a}}$ - Syntactical analysis of textual documents has been investigated severely in the last few decades. Natural language processing is the main technique used to extract more structural information out of documents. Efforts are directed to build models from requirements specification in natural language. The naïve approach is to use nouns as candidates for entities and verbs for relations between entities. However, there is necessity for more sophisticated techniques to handle linguistic variation when proposing model elements and constructing domain models from a large set of documents. The approach[13] of natural language analysis for semantic documents modeling is reused in our approach.

Step$_{1b}$ - Quantification of the relationship between concepts is done using linguistics and natural language processing techniques for analyzing the documents from a domain. Correlation analysis, collocation techniques, similarity thesaurus[14] are used to evaluate the strength of relationships between concepts. Computation is expensive. However, these weights have to be computed only once before starting the project. The values should be refined by the domain expert – this reflects the domain expert's belief in how much the concepts are related in a particular domain. So, these numbers come from either objective data or the experiences of the domain expert accumulated from the development of similar projects. These ranges are used to represent the high (0.7 to1.0), medium (0.4 to 0.6) and low (0.0 to 0.3) degree of relation.

Step$_2$ - Fragmentation of artifacts into semantic fragments. The produced artifact is translated to XML format and is logically fragmented according to its semantics. Fragmentation is done by a traceability module, which gets the XML file as input and where the fragment boundaries are defined by the developer. As an output, an XML file with added tags to identify start and end positions of a fragment is produced.

Step$_3$ - Fragments mapping to the concepts. Candidate concepts to build a domain model are suggested automatically by the processing of the fragments. Techniques from **Step$_1$** are adapted to extract the concepts, if possible, from the fragments and to propose the closest related concept from the domain model to map to it. Fragments can be linked directly to other fragments if developer finds them related or if one fragment is a part of another. Further, a fragment can be linked to a domain concept, see the meta-model description below for more detailed explanation. The weighting scheme is similar to the one described in ***Step$_{1b}$***. The mapping rate is revised and confirmed by the developer, who created new or a version of the fragment and checked-in to the repository. The relationship information is encoded using XML tags. Finally, the fragments are stored in a central repository.

The domain model is constructed and the concepts are related with weighted links according the strength of the concept relations. The weights are then used to evaluate relations between fragments when mapped to the domain concepts and to estimate the likelihood of impact of one fragment on another.

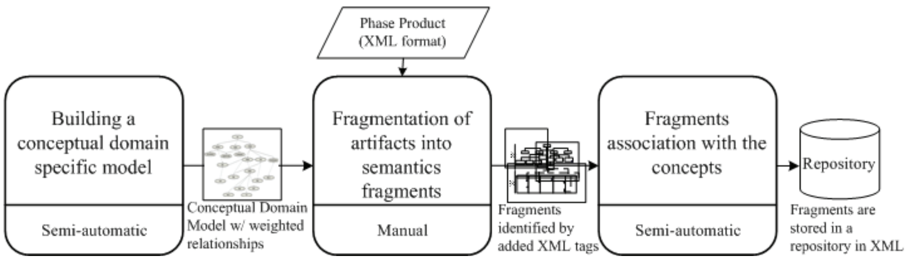

Phase Product (XML format)
Building a conceptual domain specific model
Semi-automatic
Conceptual Domain Model w/ weighted relationships
Fragmentation of artifacts into semantics fragments
Manual
Fragments identified by added XML tags
Fragments association with the concepts
Semi-automatic
Repository
Fragments are stored in a repository in XML

Figure 1. Main steps to enable change impact assessment

Thus, dependency relations are based on the semantics of the artifacts. Fragments are linked to the concepts from the domain model; all selected fragments are mapped and linked through the conceptual domain model as follows.

There exists a set of concepts {C_1, C_2, …, C_n} and a set of fragments {F_1, F_2, …, F_m}, then consequently:

– If fragment F_i is mapped to a concept C_i and fragment F_j is mapped to C_i, then transitively F_i also relates to F_j (Eq.1).

$$(\mathtt{F_i} \rightarrow \mathtt{C_i}) \wedge (\mathtt{C_i} \rightarrow \mathtt{F_j}) \Rightarrow \mathtt{F_i} \rightarrow \mathtt{F_j} \tag{1}$$

– Given, the related concepts C_i and C_j, and if fragment F_i is linked to a concept C_i and a fragment F_j is linked to C_j, then dependency to certain degree exists between F_i and F_j (Eq.2).

$$(\mathtt{C_i} \rightarrow \mathtt{C_j}) \wedge (\mathtt{F_i} \rightarrow \mathtt{C_i}) \wedge (\mathtt{C_j} \rightarrow \mathtt{F_j}) \Rightarrow \mathtt{F_i} \rightarrow \mathtt{F_j} \tag{2}$$

The meta-model for the proposed approach, based on the settings above, is depicted in figure 2 using RML (Referent Model Language)[15]. RML is an EER-like (Extended Entity Relationship) language with strong abstraction mechanism and sound formal basis.

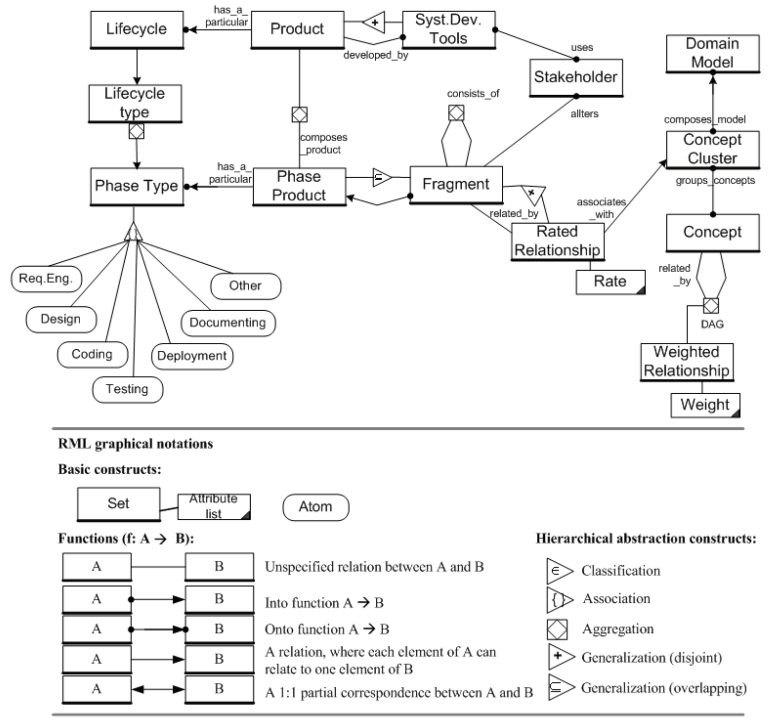


Figure 2. Meta-model to relate product fragments through the conceptual domain model

Meta-model describes the scope of the approach. We deal with `product` development, using system development tools (`Syst.Dev.Tool`), system development tool can be also seen as `product`, when it is under development. Every product development has specific `lifecycle` consisting of different `phase type` (e.g., business analysis, requirements engineering, design, implementation, testing, etc.). Each `phase type` has a distinct `phase product type` (e.g. requirements specification, design, code, user manual, and software itself), which is result of particular lifecycle phase. A `product` is final result of the development project, and it consists of the interrelated `phase product type`.

A `fragment` is a semantic piece of `phase product type` in a certain granularity level, e.g., it can be a document, a model, a diagram, a section in a document, a text specifying a non-functional requirement, an use case, a class, an attribute, etc. `Fragment` can consist of fragments. It should be noted that `fragment` is inreflexive, asymmetric, and non-transitive. `Fragment` can have a direct dependence link to another `fragment`. Every `fragment` has semantics, which relate the `fragment` to one or more `concept cluster`. `Rated` mapping `relationship` is used to distinguish `fragment` coherency to a particular `concept`. `Concept cluster` groups related concepts and composes `domain model`. `Concept` is connected to other concept by direct acyclic graph with weights (`weighted relationship`). Weights of those relations are calculated based on degree of the concept relatedness.

4. APPLICATION OF THE APPROACH

In this section we present a case example to test practical applicability and illustrate the proposed approach in empirical settings. Description of the application of the approach consists of a realistic case of and a candidate technique for quantitative analysis – Bayesian Belief Network.

4.1. A Case Study

A case study is based on MEIS (Model Evaluation Information System) system, used for the basic course of information systems SIF8035[16]. MEIS system is used for exercise delivery and evaluation. There are two groups of users: students that are also reviewers of others' solutions, and student assistants, who check all deliveries including both solutions to an exercise and evaluation of those solutions and either accept or reject them. The domain concepts in exercise delivery and evaluation and the relationships among them are depicted in figure 3. Here, the quantification of the relationships between concepts has been performed manually based on our knowledge of the domain. For example, the weight of relationship between '`Student`' and '`Reviewer`' is equal to `1.0` only in this domain, where students are also reviewers of others' solutions.

Next, during the development of the MEIS system every requirement was treated as a separate fragment. Some of them are listed below and additional examples of the product fragments are presented in figure 4. Requirements for the MEIS system are[16]:

Req.1. It should be possible to create users' profiles from a textual file.
Req.2. A student should be able to upload a solution:
  Req.2.1. A solution should be stored in the student's folder.
  Req.2.2. A reference (link) to a solution1/2 should be kept in the MEIS database.
Req.3. StudAssist should accept/reject a solution1/2.

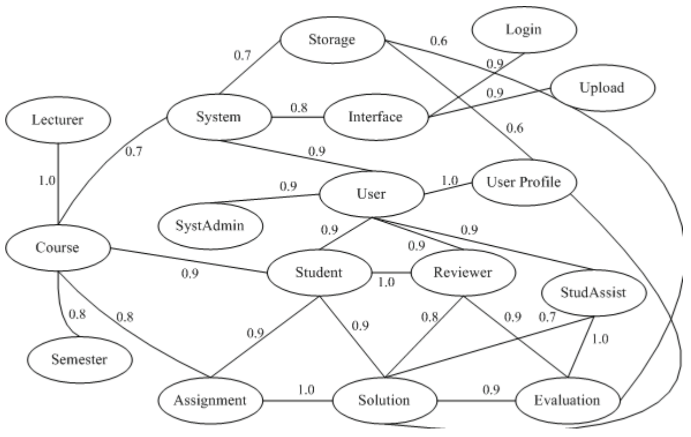


Figure 3. Domain model for MEIS

Req.3.1. The system should provide the possibility to reject a solution1/2.

Req.4. StudAssist should form a reviewer groups for a solution1/2.

Req.4.1. The system should provide StudAssist a list of students, whose solution has been accepted.

Req.4.2. StudAssist should form a reviewer group based on the student list in Req.4.1.

Req.5. Reviewer should deliver evaluations of both the solution1 and solution2.

Req.5.1. Reviewer should evaluate the DFD/APM model of the solution1/2.

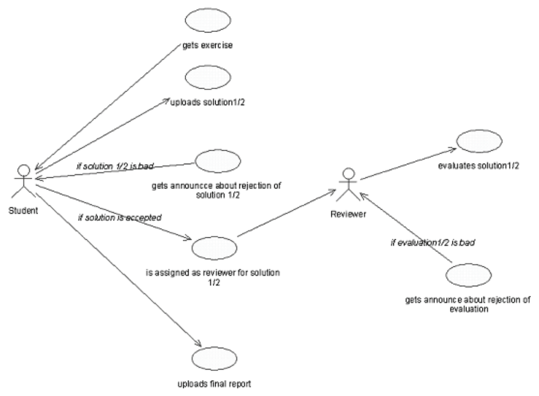
Req.5.2. Reviewer should upload Word documents with evaluation for the DFD/APM model of the solution1/2.

Req.5.3. File with evaluation for the DFD/APM model of the solution1/2 should be stored in the database.

As described in the previous section, developers will be provided with the tool for semi-automatic fragments mapping to domain concepts. Additional XML tags are entered to keep information about the related concepts and the weight of their relationships, as a fragment could have one or more related concepts. For example, requirement "Req.5.2: Reviewer should upload Word documents with evaluation for the DFD/APM model of the solution1/2" provides hints about the relation to the concepts '`Reviewer`', '`Upload`', '`Evaluation`' and '`Solution`'. Nevertheless, the requirement is mainly about 'evaluation upload'. Therefore, it is mapped to the concepts '`Upload`' and '`Evaluation`' with the assigned weights `0.9` and `0.7`, respectively. A partial graphical representation of the fragments as mapped to the domain model is depicted in figure 5. The concepts and fragments from the example above are gray shaded. It should be noted that figure 5 is not intended normative for fragments mapping, but is used here only for illustrational purposes.

4.2. Bayesian Belief Network

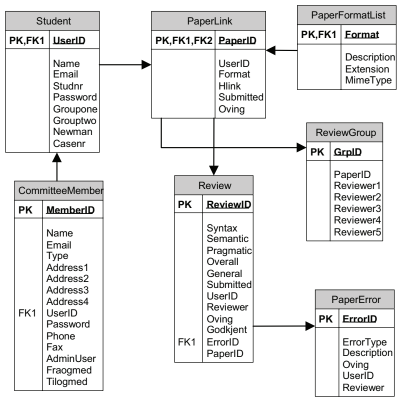
One candidate for implementing the approach for product traceability is Bayesian Belief Network (BBN, also called Bayesian Network or Probabilistic Networks). BBN is a powerful technique for reasoning under uncertainty[17, 18] and representing knowledge. It provides a graphical model that resembles human reasoning. In the recent decades, Bayesian Belief Network has attracted attention from both the research and industrial



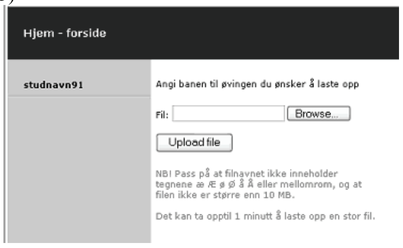
(a) Use Case diagram – students tasks ('UC.1' in fig.5)

```
if ((dbproc = mysql_init(NULL)) == NULL) {
    printf("Unable to init.\n<hr>");
  } else {
    if (mysql_real_connect(dbproc, NULL,
"ADMIN_USER",
                           "ADMIN_PASSWORD",
"DATABASE_NAME",
                           0, "MYSQL_SOCK",
0) == NULL) {
      printf("Unable to connect.\n<hr>");
    } else {
      if (is_modify) {
        passwd = Find_Value(entries,
num_words, "Password1");
        if (passwd == NULL || strlen(passwd)
<= 0) {
          sprintf(passwd_buf, "'%s'",
                  Find_Value(entries,
num_words, "Password"));
        } else {
          sprintf(passwd_buf,
"PASSWORD('%s')", passwd);
        }
```

(b) part of code ('Code.1' in fig.5)



(c) ER diagram of MEIS database ('Dsgn.1' in fig.5)


Hjem - forside

studnavn91

Angi banen til øvingen du ønsker å laste opp

Fil:

NB! Pass på at filnavnet ikke inneholder tegnene æ Æ ø Ø å Å eller mellomrom, og at filen ikke er større enn 10 MB.

Det kan ta opptil 1 minutt å laste opp en stor fil.

(d) Interface screenshot ('Doc.1' in fig.5)

Figure 4. Examples of fragments[16]

communities. BBN provides a natural way to structure information about a domain. One advantage of the BBN is that it not only captures the qualitative relationships among variables (denoted by nodes) but also quantifies the conceptual relationships. This is achieved by assigning a conditional probability to each node in the BBN[‡].

In a BBN, for each variable `x` with parent `Parent(x)`, there is a corresponding conditional probability distribution `P(x|Parent(x))`. For example, in the MEIS domain, the probability of having an impact on requirement '`Req.5.1`' is directly conditioned by the relation of the two concepts '`Reviewer`' and '`Solution`' - '`Req.5.1`' is mapped directly to them, with the concept which has mapped the changed fragment. Thus, the conditional probability is given as `P(Impact.Req.5.1 | Reviewer, Solution)`.

[‡] This has also been the main disadvantage of the BBN – human labor intensity and domain expert dependency.

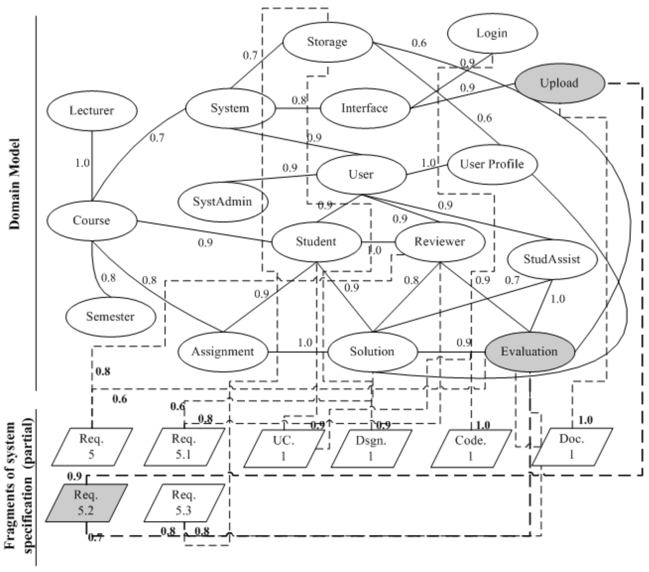


Figure 5. Graphical representation of MEIS fragments mapping to domain concepts (partial)

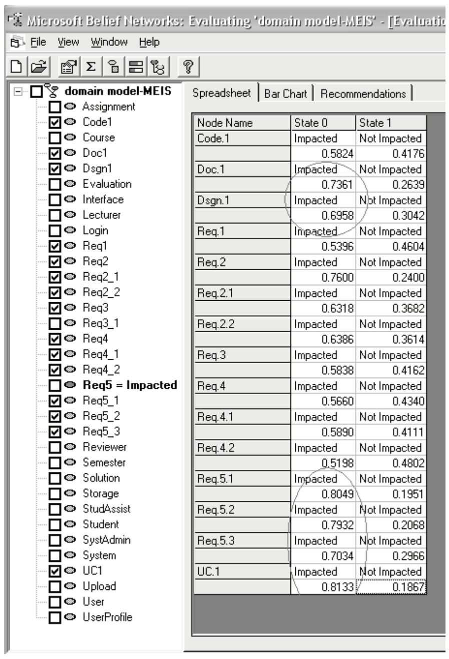
The applicability of BBN is demonstrated by continuing the example in the sub-section above and using tool MSBNx (Microsoft Bayesian Network Editor and Tool Kit)[19]. MSBNx tool was chosen as it offers an extensive COM-based API for editing and evaluating Bayesian Networks.

So, during the system development, the stakeholder decided to create a standard web form for evaluation instead of delivering an evaluation in a Word file. As a consequence, requirement (‘`Req.5`’) has been changed to:

Req.5. Reviewer should be provided 2 (two) web evaluation forms for each solution1/2.

Assessments of the impact probabilities on the other artifacts caused by this change are shown in figure 6. As all mapped fragments are assessed, only the ones with the highest probability of impact should be checked. Of course, a probability threshold should be defined for notification posting. Further, in large development projects, the threshold value depends on specific project settings and requires attentive empirical study. The definition of the impact relations between phase products facilitates the management of the calculated change impact probability values. In this case, when altering the requirement, only possible changes in the requirements specification and the related phase (e.g., design) should be notified and checked first. Change notification should proceed only if the related design fragment is found impacted, where the developer will need to check the next phases in the impact chain.

The values in figure 6 show for example, that most likely the fragments ‘`UC.1`’, ‘`Req.5.1`’ and ‘`Req.5.2`’ are impacted. The changed requirement ‘`Req.5`’ is mapped to the two concepts ‘`Reviewer`’ and ‘`Evaluation`’. As ‘`UC.1`’ is mapped to the concepts


Microsoft Belief Networks: Evaluating 'domain model-MEIS' - [Evaluatic
File View Window Help
Spreadsheet
Bar Chart
Recommendations

- ☐ domain model-MEIS
  - ☐ Assignment
  - ✓ Code1
  - ☐ Course
  - ✓ Doc1
  - ✓ Dsgn1
  - ☐ Evaluation
  - ☐ Interface
  - ☐ Lecturer
  - ☐ Login
  - ✓ Req1
  - ✓ Req2
  - ✓ Req2_1
  - ✓ Req2_2
  - ✓ Req3
  - ☐ Req3_1
  - ✓ Req4
  - ✓ Req4_1
  - ✓ Req4_2
  - ☐ **Req5 = Impacted**
  - ✓ Req5_1
  - ✓ Req5_2
  - ✓ Req5_3
  - ☐ Reviewer
  - ☐ Semester
  - ☐ Solution
  - ☐ Storage
  - ☐ StudAssist
  - ☐ Student
  - ☐ SystAdmin
  - ☐ System
  - ✓ UC1
  - ☐ Upload
  - ☐ User
  - ☐ UserProfile

Node Name	State 0	State 1
Code.1	Impacted	Not Impacted
	0.5824	0.4176
Doc.1	Impacted	Not Impacted
	0.7361	0.2639
Dsgn.1	Impacted	Not Impacted
	0.6958	0.3042
Req.1	Impacted	Not Impacted
	0.5396	0.4604
Req.2	Impacted	Not Impacted
	0.7600	0.2400
Req.2.1	Impacted	Not Impacted
	0.6318	0.3682
Req.2.2	Impacted	Not Impacted
	0.6386	0.3614
Req.3	Impacted	Not Impacted
	0.5838	0.4162
Req.4	Impacted	Not Impacted
	0.5660	0.4340
Req.4.1	Impacted	Not Impacted
	0.5890	0.4111
Req.4.2	Impacted	Not Impacted
	0.5198	0.4802
Req.5.1	Impacted	Not Impacted
	0.8049	0.1951
Req.5.2	Impacted	Not Impacted
	0.7932	0.2068
Req.5.3	Impacted	Not Impacted
	0.7034	0.2966
UC.1	Impacted	Not Impacted
	0.8133	0.1867

Figure 6. Impact probability evaluated by using MSBNx

'`Student`', which is strongly (`1.0`) related to '`Reviewer`' in this domain, and '`Evaluation`', the probability to be impacted is relatively high.

5. DISCUSSION AND CONCLUDING REMARKS

In this paper, a methodological approach to facilitate product change management in the distributed development projects has been described. The proposal is based on semantic enrichment of produced fragments by mapping them to related concepts from a specific domain model. This information is used to abstract away from heterogeneous representation details and to capture information content. In this way, domain specific conceptual model is used to interoperate across different representation formats used in the system development. Further, the relations between the fragments and concept as well as the relations among the concepts in the domain model are weighted. Weights assigned to relationships are used as the basis for impact prediction and assessment.

The contribution of this work is threefold. First, unlike other approaches, the proposed approach (a) covers whole lifecycle. Second, as the nature of collaborative development is usually highly iterative, the approach (b) supports the relating and interchanging fragments of a product at different stages of its incremental refinement (e.g., from abstract sketches to a formal representation, see **$\mathbf{Req_3}$**). Third, it (c) does not bind the developer to some specific tool and/or modeling language (see **$\mathbf{Req_1}$** and **$\mathbf{Req_2}$**) provided the preferred tool supports XML format. The use of XML enables use of this approach in settings where the involved artifacts are created and managed by heterogeneous tools, such as text processors and CASE-tools.

The approach can be beneficial for companies working in specific domains where typically the domain model is stable and commonly agreed, and an expert's knowledge is available. In the case of entering a new domain, the company should work out a specific domain model, which needs to be comprehensible and agreed by all developers.

An evolving domain model is a challenge, which should be resolved in future work. The adding or removal of some concepts from a conceptual domain model in the middle of a project raises the question of what to do with the fragments which have been mapped to the concepts. If a new concept is added, the relatedness between the concept and closest fragments could be automatically calculated and the most related fragments could be re-mapped. A concept deletion should not remove the concept from the domain model, but should lock it order to prevent mapping to any new fragments. This would preserve existing links between the concepts and fragments.

Further, large domain model with thousands of concepts could be a challenge for developers in finding the relevant concepts and to link the fragment in question. A candidate solution here is concepts clustering, which could facilitate the selection of the right concept. Developments in the area of ontology mapping could also provide useful methods and techniques which could be used both to find the most relevant concept for the fragment and to develop a stable and a commonly agreed domain specific model for new domain, where several interpretations of the domain and the model exist.

However, where the main contribution of this paper is in change impact assessment as being vital for large development projects, simultaneously, it is perhaps the most risky and error-prone task. The proposed approach enables to calculate the probability objectively from subjective materials – how likely some product fragments are to be impacted by a change of a 'semantically related' fragment. The probability value is calculated based on the weighted relations between domain concepts, where the weights depend on experts' knowledge of the domain. As the calculation operating on those weights is the backbone of this approach, the process of weight assignment should be well reasoned and methodologically described as well as empirically tested - big challenges for future work lie here.

6. ACKNOWLEDGMENTS

Special thanks go to Sobah Abbas Petersen and the anonymous reviewers for proof reading. This work has been partially supported by the Simula foundation in Norway.

7. REFERENCES

1. S. Greenspan, C. McGowan, Structuring Software Development for Reliability, In *Microelectronics and Reliability*, 17, 1978, pp. 75–84.
2. O.C.Z. Gotel, A.C.W. Finkelstein, "An Analysis of the Requirements Traceability Problem", In *Proceeding of the 1st International Conference on Requirements Engineering* (ICRE'94), IEEE Computer Society Press, Colorado Springs, Colorado, USA, April 1994, pp. 94–102.
3. R. Watkins, M. Neal, "Why and How of Requirements Tracing", *IEEE Software,* 1994 11(4), pp. 104-106.
4. D. Strašunskas, "Traceability in a Collaborative Systems Development from Lifecycle Perspective", in *Proceedings of the 1st International Workshop on Traceability*, co-located with ASE 2002, Edinburgh, Scotland, UK, September 2002, pp. 54-60

5. F. Pinheiro and J. Goguen. "An Object-Oriented Tool for Tracing Requirements". *IEEE Software*, 1996 13(2), pp. 52-64.
6. K. Pohl, "PRO-ART: Enabling Requirements Pre-Traceability", In *Proceedings of the Second International Conference on Requirements Engineering* (ICSE '96), Colorado, USA, 1996, pp. 76-85.
7. P. Grünbacher, A. Egyed, and N. Medvidovic, "Reconciling Software Requirements and Architectures - The CBSP Approach", In *Proceedings of the 5th IEEE International Symposium on Requirements Engineering* (RE'01), Springer-Verlag, Toronto, Canada, 2001, pp. 202-211.
8. K. Pohl, M. Brandenburg, A. Gülich, "Integrating Requirement and Architecture Information: A Scenario and Meta-Model Based Approach", In *Proceedings of the Seventh International Workshop on Requirements Engineering: Foundation for Software Quality* (REFSQ'01), Interlaken, Switzerland, 2001.
9. P. Letelier, "A framework for Requirements Traceability in UML based projects", in *Proceedings of the 1st International Workshop on Traceability*, co-located with ASE 2002, Edinburgh, Scotland, UK, September 2002, pp. 32-41.
10. S.T. Frezza, S.P. Levitan, P.K. Chrysanthis, "Linking requirements and design data for automated functional evaluation", *Computers in Industry*, Volume 30, Issue 1, Elsevier Science Publishers B. V., September 1996, pp. 13-25.
11. A. Egyed, "Reasonings about Trace dependencies in a Multi-Dimensional Space", in *Proceedings of the 1st International Workshop on Traceability*, co-located with ASE 2002, Edinburgh, Scotland, UK, September 2002, pp. 42-45
12. B.A. Farshchian, *A Framework for Supporting Shared Interaction in Distributed Product Development Projects*, PhD thesis, NTNU, Trondheim, Norway, 2001.
13. T. Brasethvik and J.A. Gulla. "Natural Language Analysis for Semantic Document Modeling." In *Proceedings of the 5th International Conference on the Application of Natural Language for Information Systems* (NLDB'2000) in Versailles, France, June 2000
14. R. Baeza-Yates and B. Ribeiro, *Modern Information Retrieval*. Addison-Wesley, 1999
15. A. Sølvberg and T. Brasethvik, "The Referent Model Language", Technical Report. NTNU, Trondheim, Norway; http://www.idi.ntnu.no/~ppp/referent/
16. R. Matulevičius, "MEIS requirements specification", Technical report, NTNU, Trondheim, NTNU, June 2003
17. F.V. Jensen, *An Introduction to Bayesian Networks*. UCL Press, London. 1996.
18. J. Pearl, Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference, Morgan Kaufmann. 1988.
19. Microsoft Bayesian Network Editor and Tool Kit; http://research.microsoft.com/adapt/MSBNx/

F.5 Model and Knowledge Management in Distributed Development: Agreement based Approach

Strasunskas, D., and Lin, Y. Model and knowledge management in distributed development: agreement based approach. In Vasilecas, O. et al. (Eds.), *Information Systems Development: Advances in Theory, Practice, and Education*[24], Springer, 2005, pages 389-402.

[24] Proceedings of the 13th International Conference on Information Systems Development (ISD 2004), Vilnius, Lithuania, September 2004.

MODEL AND KNOWLEDGE MANAGEMENT IN DISTRIBUTED DEVELOPMENT: AGREEMENT BASED APPROACH

Darijus Strašunskas and Yun Lin*

1. INTRODUCTION

Models are built to share knowledge or definitions with other people, and this application is especially directed to people that want to share knowledge or define knowledge in co-operation with others. Modeling is seen as "the activity of formally describing some aspects of the physical and social world around us for purposes of understanding and communication"[1], and is applied in the early phases of information system analysis and design. However, many problems are encountered when building models. It is conceivable that a variety of different versions of models will be used in different stages of the development process; in general, it is difficult to develop a model that can be acceptable for all participants in a development project. Furthermore, it is known that different people usually present different models given the same domain and the same problem. The same information about system can be modeled at various levels of abstraction and from different viewpoints considering different aspects. Variations among models generally appear due to the creative nature of the modeling activity, as well as other factors such as the richness of the modeling language[2], the ambiguities of modeling grammars, and others.

This problem becomes more evident when the process is distributed. Then the variability of the model versions increases due to the highly interactive and iterative nature of the development process and to the different, sometimes conflicting, angles to the problem and solution taken by the different stakeholders. Therefore, modeling process can be viewed as three dimensions of requirements engineering[3]: agreement, representation and specification dimension. The agreement dimension should be based on common understanding about problem domain, organizational strategy; the representation dimension is based on the essential semantic aspects of system analysis; the specification dimension bases on the implementation oriented system development aspects. The most difficult in

* Darijus Strašunskas, Dept. of Computer and Information Science, Norwegian Univ. of Science and Technology, NO-7491 Trondheim, Norway and Department of Informatics, dstrasun@idi.ntnu.no.
Yun Lin, Dept. of Computer and Information Science, Norwegian Univ. of Science and Technology, NO-7491 Trondheim, Norway; yunl@idi.ntnu.no.

modelling is to arrive at a coherent, complete and consistent description of the problem and domain. Description should be shared and agreed between all stakeholders. Therefore, we focus on the agreement dimension that deals with the consolidation of logically and geographically distributed views[3].

In this paper, we discuss distributed modeling, focusing on support for individual developers and allowing them expressing their view and perception of the Universe of Discourse (UoD) in model fragments, which are integrated based on common agreement among them. The paper is further structured as follows. Section 2 overviews related work. In section 3, we further elaborate complexity of distributed modeling, discuss settings for our approach and present the approach. In section, 4, we illustrate our approach using example from a travel domain. Finally, section 5 concludes the paper and lays down future work.

2. RELATED WORK

Many proposals on model composition are available in the literature. Model composition in a distributed heterogeneous environment has been the subject of a few recent research activities. Namely, the merging of ontologies is one of the recent model merging scenario. Collaboration during the modeling is one of the most important features of the ontology building tools, as ontology is seen as an explicit representation of a *shared* conceptualization[4]. However, less than half of ontology building tools surveyed in[5] have a multi-user support. Even the tools supporting collaborative ontology development still do it in the old-fashioned way, i.e. do not allow multiple accesses to concept (object) by different developers at the same time. The work that has been done so far in the area of collaborative work with ontologies mainly has focused on one ontology which is edited by the developer. I.e., the web-based Ontosaurus[†] supports collaboration and allow developers to edit ontology only when consistency is retained within the ontology as a whole.

Environments like Protégé[‡] or Chimaera[§] offer sophisticated support for ontology engineering and merging of ontologies, but lack sophisticated support for collaborative engineering. Chimaera is build on top of Ontolingua Server[6] and, therefore, has the same support for collaborative engineering, i.e. read and write access rights to ontologies are controlled by the ontology owner; users are able to join a session and work simultaneously on the same ontology.

Some tools provide advanced support for communication between users contributing to better collaboration during ontology engineering, e.g. Tadzebao[7] supports both asynchronous and synchronous discussions on ontologies; Apecks[8] aims to support discussion about ontologies and allows different conceptualizations of a domain to co-exist.

In[9] we found the first attempt to use a totally distributed environment to work with ontologies. They present their work with the peer-to-peer Semantic Web. It allows users to create, maintain, and control sharing of ontologies in a P2P environment. Although it allows users to add parts to ontologies, but it mainly seems to be built for maintaining, sharing and retrieving other ontologies.

WebOnto[**] is a web-based tool for developing and maintaining ontologies. It in-

[†] http://www.isi.edu/isd/ontosaurus.html
[‡] http://protege.stanford.edu/
[§] http://www.ksl.stanford.edu/software/chimaera/
[**] http://webonto.open.ac.uk/

cludes functions such as visualization, browsing and editing ontologies. The tool includes functionality for sharing changes between users. Mintra et al.[10] present a toolkit called Onion. It is a toolkit to help domain expert bridge the gap between smaller domain specific ontologies by creating links between ontologies based on their context. Before Onion was developed most research on ontology construction focused on tools for building a single global ontology.

Hozo[11] environment for distributed ontology development is focused on building a single ontology by on splitting it into components and establishing dependency between them. The target ontology is obtained by compiling the component ontologies. System does not allow multiple accesses to a concept by different developers at the same time, as developers have assigned a particular component ontology to develop. OntoEdit[12] allows multiple user access to the same ontology to build it collaboratively. It provides namespace mechanism allowing splitting ontologies into modules.

In summary, most tools provide the collaborative facilities by supporting basic requirements for distributed development, e.g. rights- and user management, locking mechanism, communication and notification means. Even most sophisticated modeling environments do not provide means for development of the *shared conceptualization*, i.e. allowing users to develop overlapping fragments of models based on their own understanding and perception of the real world.

3. DISTRIBUTED MODELING

3.1. Complexity of distributed modeling

Model development is a complex and difficult task. It is usually a creative and collaborative process, during which different stakeholders are focusing on various aspects, expressing them at different levels of abstraction, and producing several variants of each.

Different levels of abstraction of the same system enable to deal with complexity by removing details from model. The model must be able to act as an efficient and effective communications medium between the different parties involved in development project. Usually, models are augmented by details on each development step.

A system can be described from many viewpoints. Each viewpoint defines what characteristics should be included in its views and what issues should be ignored or treated as transparent. A view is, therefore, a piece of the model that is small enough to comprehend but that also contains all relevant information about a particular concern. Variants dimension is more concerned with different versions and configurations.

The success of distributed project depends on how well “laissez-faire” rule is obeyed, meaning that developers should be allowed to express what they want in whatever form. More precisely, Farshchian[13] emphasized a list of requirements for development environments to enable collaboration in geographically distributed developments. Here we adopt the requirements (**Req.n**) as follows.

Req.1 - *Unrestricted product object types* – a development environment should allow the developers to share any type of object that they might find useful for supporting their cooperation.

Req.2 - *Unrestricted relation types* – a development environment should allow the developers to create any type of relation between any two objects.

Req.3 - *Incremental product refinement* – a product development environment should provide the developers with flexible mechanisms for incrementally refining the product. The developers should be allowed to start with vague products, and to refine them into more complete and formal ones.

Concurrent engineering changes old practice, when all the required objects were locked during the whole change/ modification activity. Each software engineer should have direct access to all needed objects. But changed version should be kept with access forbidden for other developers during modification, because the state of fragment is inconsistent in a modification phase. If *n* engineers change the same object concurrently, this object should have *n+1* different copies[14]. It means that each developer needs the private copies of fragments. On the other hand, the colleagues know that other changes possibly are done on the same fragments/ objects and want to be incorporated when relevant. In summary, collaborative distributed development needs tools that allow the creation and access to a central composite product, and at the same time support development in local workspaces.

3.2. Knowledge preservation

In a collaborative environment where different users work on models, it is important that there is a way of sharing own views, and step by step achieving agreement and common conceptualization. This is usually called model integration and can be accomplished by merging or term alignment. It is important to keep term merging separated from the term alignment. Merging means that one new model is created from *n* existing models. Model alignment is when links are created between models, so that the models can be used as one.

Although, the initial goal is usually to develop a single model of the UoD, it turns out to be very important to preserve and model the various “views” of the information seen by different stakeholders and participants during the system analysis phase. Usually, different developers might have different vocabulary to express their perception of the world. It is important to preserve knowledge of developer that is expressed in the model fragment she has developed. We need to ensure that developer’s work will not be disturbed, for instance, if a developer uses term ‘aircraft’ referring to ‘airplane’, this term should be preserved in her local view, otherwise after several changes it will be difficult to continue.

3.3. An approach

Underlying hypothesis of our approach is that given the same problem domain to reason about, the model developed by different stakeholders will not only differ, but as well will have some overlapping parts, i.e. some parts (views) in different models are commonly shared. In order to integrate the distributed models, these commonalities should be captured.

Our approach consists of 3 basic steps (see figure 1):

Step 1 - Model matching and similarity identification. Model integration typically involves identifying the correspondences between two models, determining the differences in definitions, and creating a new model that resolves these differences. Four types of view differences are described in[15], which were paraphrased by Hefflin and Hendler[16]:

- **terminology**: different names are used for the same concepts;
- **scope**: similar categories may not match exactly; their extensions intersect, but

each may have instances that cannot be classified under the other;
- **encoding**: the valid values for a property can be different, even different scales could be used;
- **context**: a term in one domain has a completely different meaning in another.

Some of the above listed problems might be found and resolved automatically, i.e. scope, context. At present, there exists a number of automated schema and model matching systems, for instance[17-25], which produce correspondence suggestion between elements of different models. In general, match algorithms developed by different researchers are hard to compare since most of them are not generic but tailored to a specific application domain and model types. Usually, use of only one approach will not produce good enough matching; for better results we need to combine several schema matching approaches. Currently, we are investigating which combination of techniques is most applicable for our approach.

Step 2 - "Sameness" identification. Given current techniques for correspondences assertion between models, it is possible to calculate quite precise similarity of the concepts. However, it is impossible to identify the "sameness" of concepts without knowing authors' intention. Therefore, manual intervention and agreement is necessary at this step. The authors of all model fragments are notified about the results of the model matching and are asked to verify them by pointing the same concepts and achieving agreement about the concept name, if they use different terms.

These two steps (step 1 and step 2) are iterated as many times as new fragments are signed-in to the repository. Step 2 results in the common knowledge layer or so called "concept-space", where the commonly agreed concepts and relations between them are placed. This layer is used to differentiate from the local namespace, which is kept unique for each developer allowing to use own vocabulary. Figure 2 exemplifies the idea behind our approach. I.e. after having identified the concepts being the same, despite of different term used to name them, the "equality" relationship is established between local concepts named '*A1*' and '*A2*', and the agreed concept named '*A*'.

Step 3 - composition of models. In this step a final application dependent model is produced based on agreed view and formed model in the common concept space.

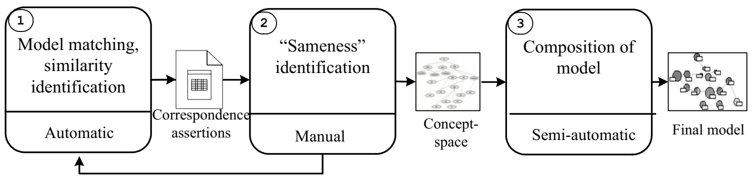

1
Model matching, similarity identification
Automatic
Correspondence assertions
2
"Sameness" identification
Manual
Concept-space
3
Composition of model
Semi-automatic
Final model

Figure 1. Functional view of our approach

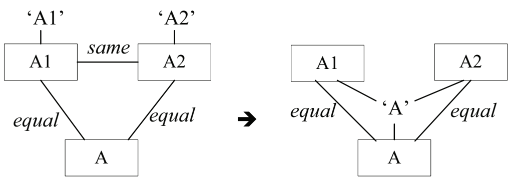


Figure 2. Management of "sameness"

3.4. Further Elaboration

Since distributed models are built by different modelers having various modeling purposes and viewpoints, the perspectives of different models may be far from each other. Context similarity is considered during the agreement and identification of the same concepts. Some constraints in different local models may conflict with each other, even being agreed and integrated in common models. Furthermore, some concepts and relationships in the integrated models may be redundant or may need to be further specified, i.e. what to do with derived relationships or model fragments at different abstraction levels. Further, we introduce more rules for model refinement (step 3).

Here, we mainly focus on static (class) diagram which presents concepts and their relationships. Hence, the integration refining issues include abstracting concepts and refining concepts, adding and deleting properties of concepts, adjusting types of properties, abstracting transitive relationships into high level relationships and refining relationships into low level relationships. We define a set of generic rules for the above mentioned refinement transformations. Before formulizing those refining issues, we make some definitions. Let UoD_I be universe of discourse for integrated model, and UoD_D – universe of discourse for particular local model fragments. Then, C_D is a concept used from a local model fragment and C_I is the concept in the integrated model. $P(c)$ is the set of properties of concept C and p is a property. While, $R(C_i{*}C_j)$ is the relationship between concepts C_i and C_j.

Rule 1. Abstraction of concepts. Concepts used in local models are usually more concrete. Often, during the integration, super concepts are needed to generalize those sub concepts, or even replace sub concepts if the sub concepts are not important in an integrated model.

Let, C_{Di} and C_{Dj} be two concepts from a model i and model j. Both concepts are elements from the same domain (UoD). Then a concept C_I from the domain of integrated model will be a super concept of C_{Di} and C_{Dj} in the integrated model.

$$C_{Di} \in UoD_I \wedge C_{Dj} \in UoD_I \wedge \exists C_I\,(C_I \in UoD_I \wedge C_{Di} \subseteq C_I \wedge C_{Dj} \subseteq C_I\,) \Rightarrow C_I = Abstract(C_{Di}, C_{Di}) \tag{1}$$

Rule 2. Refinement of concepts. There is a need to create new concepts, when UoD_I of an integrated system is broader than the one considered in the local model fragments. Some of such concepts are created based on a relationship between existing concepts.

$$R(C_{Di} * C_{Dj}\,) \in UoD_I \wedge \exists C_I(C_I \in UoD_I \wedge C_I \notin UoD_{Di} \wedge C_I \notin UoD_{Dj}\,) \Rightarrow Create(C_I, R(C_{Di} * C_{Dj}\,)) \tag{2}$$

Rule 3. Addition and/or deletion of properties of concepts. Certain properties of concepts are ignored in the distributed model fragments as being not important in a limited scope or in a certain viewpoint, but they might be critical for an integrated system. On the other hand, certain concepts contain too many details which are necessary in some isolated models, but inessential for the integrated system. Properties need to be further edited according to requirements for new integrated system.

$$\exists p(p \in UoD_I \wedge p \notin P(C_D\,)\,) \Rightarrow AddProp(p, C_I\,) \tag{3}$$

$$\exists p(p \in P(C_D\,) \wedge p \notin UoD_I\,) \Rightarrow DelProp(p, C_D\,) \tag{4}$$

Rule 4. Adjustment of types of properties. Types of properties usually concern implementation oriented aspects, and have little effects on the semantics of models. Meaning that possibly the same property has different types in different models. In order to keep the consistency of integrated model, types of the same property should be unified obeying implementation requirements of system.

Let *Sem(p)* be the semantics of property *p* and *T(p)* be the type of property *p*.

$$Sem(p_{Di}) = Sem(p_{Dj}) \wedge T(p_{Di}) \neq T(p_{Dj}) \Rightarrow Adjust(T(p_{Di}), T(p_{Dj})) \tag{5}$$

Rule 5. Abstraction of transitive relationships into higher level relationships and refinement of relationships into lower level relationships. A transitive relationship is the semantic equivalent of a collection of normal relationships[26]. The transitive abstraction relationship is the high level relationship and a direct relationship which can not be refined is low level relationship. With different requirements, perhaps only high level relationship is enough while on other cases low level relationship is necessary. There are three generic relationships – generalization, aggregation and association, which are supported by most modeling languages. The transitive abstraction rules for different combination of three generic relationships are different. In[27], they developed a set of transitive abstraction rules for inference of transitive relationships (e.g., classA-*association*-> classB<- *aggregation*-classC ⇒ classA-*weakAssociation*->classC, meaning that, if classA has association relation with classB, and classB is aggregated into classC, then the resulting abstraction would be weak association between classA and classC), which we do adopt for our purposes. Given the combination of $R(C_{Di}*C_{Dj})$ and $R(C_{Dj}*C_{Dk})$ satisfies one of transitive rules, the result would be $R(C_{Di}*C_{Dk})$, while *R* here is specified as either generalization (R_{Ge}), aggregation (R_{Ag}) or association (R_{As}) and parameters are non-transitive.

$$R(C_{Di} * C_{Dj}) \cup R(C_{Dj} * C_{Dk}) \in RuleSet \Rightarrow R(C_{Di} * C_{Dk}) \tag{6}$$

The refinement process based on rules is semi-automatic. Developers need to make decision on what concepts and what properties are important, at what kind of granularity concepts and relationships should kept as they depend on the requirements of integrated system.

4. APPLICATION OF THE APPROACH

In this section, we exemplify our approach using a case from a travel domain. There are requirements to build a travel agency system which provide airplane and train ticket, and hotel booking services as well to provide other tourism information. Separate models are made by different modelers and later they are integrated into one model. To illustrate our approach, we focus on two model fragments: airplane and train transportation model fragments. Then these two models will be integrated as a part of the whole travel agency system model.

Airplane transportation model fragment describes basic concepts and their relationships about flight. Figure 3 shows UML class diagram for airplane transportation. Train transportation model fragment contains concepts and relationships about train transportation information and is depicted in figure 4, using UML class diagram as well.

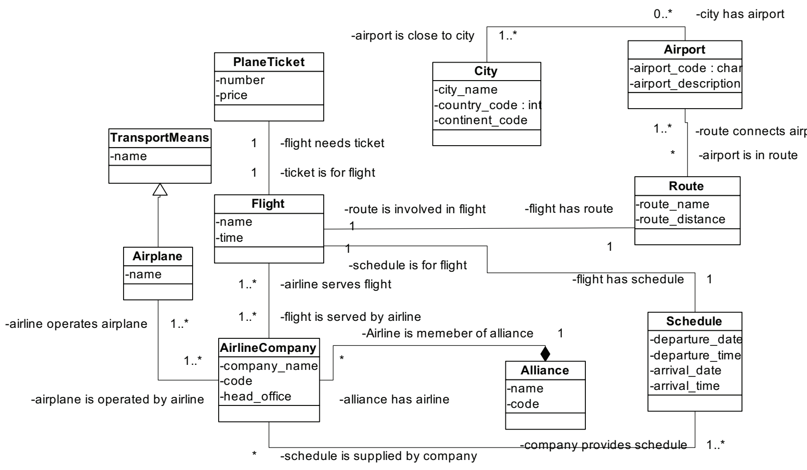


Figure 3. Airplane transportation model fragment

Step 1. Model matching and similarity identification. Because both two models are built in UML and they are quite similar in structure and in context, the model similarity can be identified by current available schema and model matching systems[17-25]. In this particular case, we have adopted iMapper[25], developed in our group. Most similar concepts pairs from the two models are {*Schedule*, *Timetable*}, {*Flight*, *Trip*}, {*City*, *City*} and {*Route*, *Route*}[††] (see figure 5).

Step 2. "Sameness" identification. With the list of similar concept pairs, modelers should reach agreements on whether two concepts are same or not (see figure 5). Concepts *'Schedule'* and *'Timetable'* are regarded as being the same, only different terminology used. '*Timetable'* is decided as a common concept name for this concept, so *'Timetable'* is put in the common concept-space and is referred by *'Schedule'* in airplane transportation model fragment and by *'Timetable'* in train transportation model fragment. *'Trip'* is put in the common concept-space as the reference of *'Flight'*. Trip is chosen because the name of '*Flight'* is more specified to airplane but the structure of it is same as *'Trip'* concept. Two *'Routes'* concepts are regarded as the same. Two *'City'* concepts look almost the same, but the type of property '*country_code*' in two models are different: one is *'int'* and the other is *'char'*. Such difference is kept in common concept space and will be resolved in step 3 as it depends on an application.

Step 3. Composition of model. As *'AirlineCompany'* and *'RailwayCompany'* refer to different entities, the way to integrate them is by generalizing and relating them by more abstract concept. Therefore, we apply Rule 1 (see Eq. 1):

$$\begin{aligned}&(AirlineCompany \in TravelDomain) \wedge (RailwayCompany \in TravelDomain\ ) \wedge \\ &(TransportCompany \in TravelDomain) \wedge (AirlineCompany \sqsubseteq TransportCompany) \wedge \\ &(RailwayCompany \sqsubseteq TransportCompany)\end{aligned} \tag{7}$$

[††] The first concept in parentheses is from airplane transportation ontology model fragment and the second one is from train transportation ontology model fragment.

And as result we introduce an abstract concept *'TransportCompany'* being the super concept of *'AirlineCompany'* and *'RailwayCompany'*.

The generation links between *'AirlineCompany'*, *'RailwayCompany'* and *'TransportCompany'* should be added into integrated model. When the generation links are added in the model, other relationships related to *'AirlineCompany'* and *'RailwayCompany'* should be checked if they are consistent with *'AirlineCompany'* and *'RailwayCompany'* or link them directly to their super concept *'TransportCompany'*.

Applying Rule 4 (Eq. 5): Concept *'City'* in figure 3 is considered the same as in figure 4, but the type of property '*country_code*' in figure 3 is *'int'* while in figure 4 it is *'char'*. Type *'int'* is adjusted into *'char'* in the integrated model.

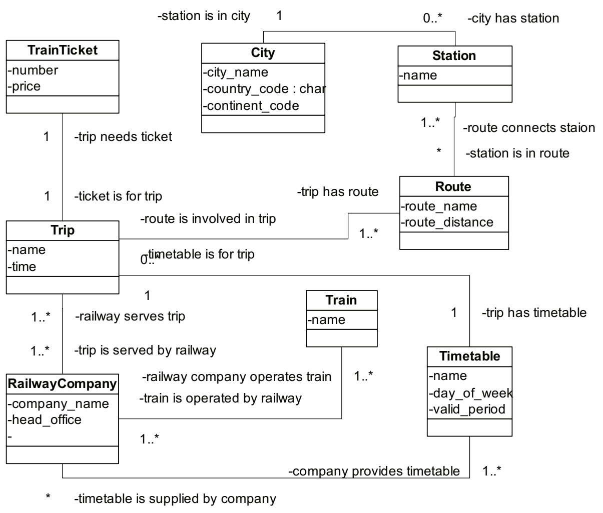


Figure 4. Train transportation model fragment

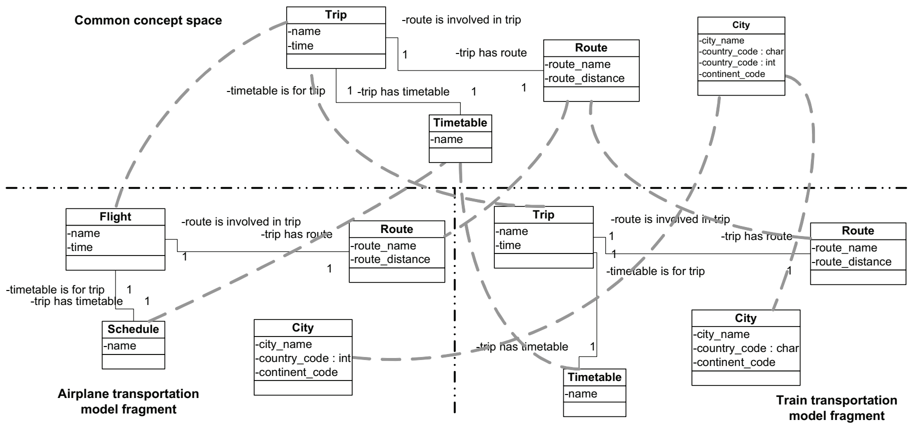


Figure 5. Explanatory visualization of mappings between models in local namespaces (bottom part) and concept space (upper part).

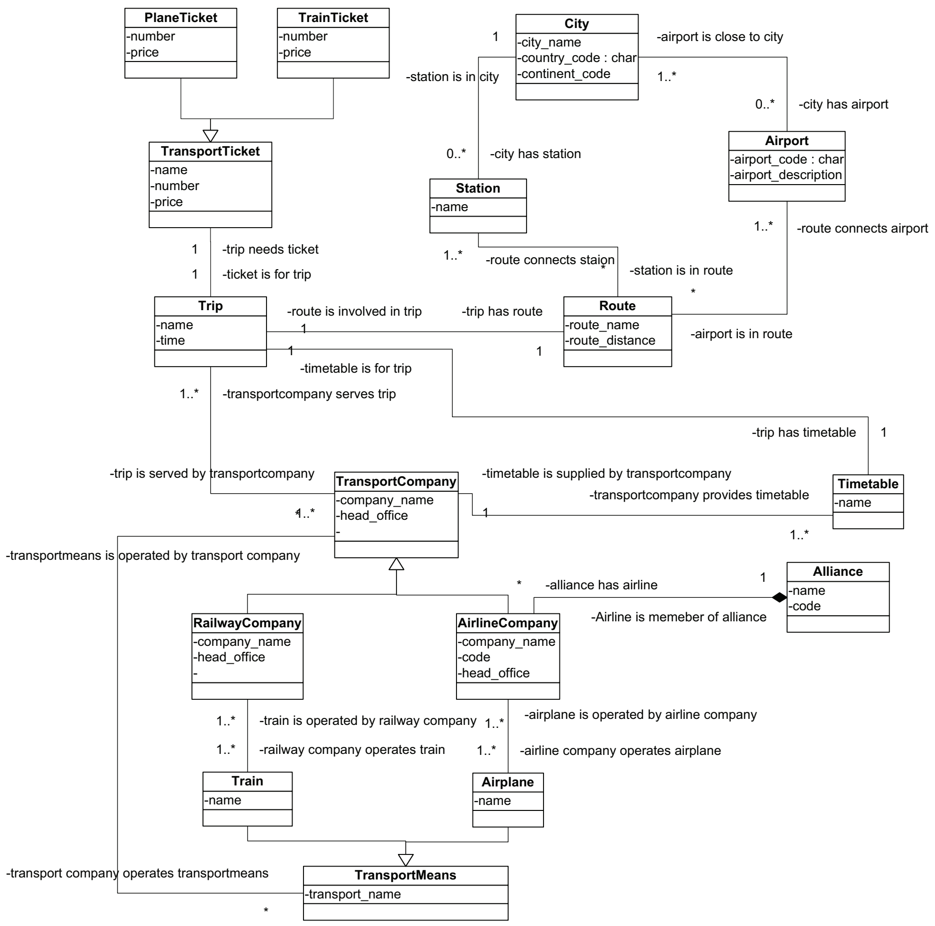


Figure 6. Integrated model for travel domain

Applying Rule 5 (Eq. 6): *'TransportTicket'* is inserted as super concept (Rule 1) of *'PlaneTicket'* (figure 3) and *'TrainTicket'* (figure 4). *'PlaneTicket'* has a relationship with concept *'Flight'*, as well as *'TrainTicket'* is related to concept *'Trip'*, and during sameness check we have agreed on that *'Flight'* is same as *'Trip'*. When integrating models, we should remove all the relationships connected with *'Flight'* to *'Trip'*. The link between *'PlaneTicket'* and *'Flight'* is removed and changed into relationship between *'PlaneTicket'* and *'Trip'*. Such relationship is semantically equivalent to the one between *'TrainTicket'* and *'Trip'*. The two relationships could be abstracted to the relationship between *'Trip'* and *'TransportTicket'* because of transitive relationship rule:

$$R_{Ge}(TransportTicket * PlaneTicket) \cup R_{As}(PlaneTicket * Trip) \Rightarrow R_{As}(TransportTicket * Trip) \qquad (8)$$

Finally, the integrated model is shown in figure 6. It is based on the concepts identified being the same (i.e., figure 5) and the rules for model refinement. It should be noted that the local distributed model fragments are still kept unchanged.

5. CONCLUSIONS AND FUTURE WORK

We have outlined a framework to support management of distributed modeling activities by distinguishing 2 main layers: the local namespace; the shared and agreed concept space. The local namespace allows the developers to model their views as they perceive and use their preferable terminology, i.e. providing full "laissez-faire" for their creativity. The concept space is used for sharing of conceptualization. The concept-space, or common knowledge layer, results into the target model. The most important contribution of this paper is that separation between these two "conceptual" spaces provides means for preserving knowledge of each developer, allowing them to use their own terminology, agree about individual conceptualization and still refer to the common concept space for agreement purposes.

Model fragments alignment in the concept space provides a good means for learning while modeling. For instance, having identified parts of models being the same, the developers are notified about certain mismatches (i.e. class concepts are the same, but relationship type between them differs), and discuss the difference. Therefore, we consider it being important to stepwise integrate model fragments during the development time, not only the products (models) themselves. That is the main difference from current state-of-the-art, where existing methodologies focus on models, as final product, integration.

The approach has limitation as it has not been yet tested in real distributed settings. Having different people involved it may be difficult for them to agree about whether some concepts are the same. But we believe that model integration during the development activities having authors present will produce better results, than any other post-development based integration, when authors of constituent model fragments are not available.

The approach is first step towards implementing the environment for collaborative modeling considering other aspects of collaboration, e.g. user awareness, support for opportunistic communication. Future work mainly concerns developing mechanism for recording all operations performed, tracing the information and decisions based on which concepts were added into the common knowledge layer. The challenge is creating an algorithm for automatic update of the models in the concept-space based on observed changes in the local model fragments.

6. REFERENCES

1. J. Mylopoulos, Conceptual modeling and Telos. Chapter 2 in: *Conceptual Modeling, Databases, and CASE*, edited by P. Loucopoulos and R. Zicari, Wiley, (1992), pp. 49-68.
2. T. Moriarty, The importance of names, *The Data Administration Newsletter* **15**, (2000).
3. K. Pohl, The three dimensions of requirements engineering, in: *proceedings of 5th Intl. Conf. on Advanced Information Systems Engineering* (CAiSE'93), edited by C. Rolland, F. Bodart, and C. Cauvet, Springer-Verlag, Paris, France, (1993), pp. 275-292.
4. T.R. Gruber, A translation approach to portable ontology specifications, *Knowledge Acquisition*, **5**(2), (1993).
5. M. Denny, Ontology Building: A Survey of Editing Tools (2002); http://www.xml.com/lpt/a/2002/11/06/ontologies.html.

6. A. Farquhar, R. Fikes, and J. Rice, The Ontolingua server: a tool for collaborative ontology construction, KSL Stanford University, USA (1996).
7. J. Domingue, Tadzebao and WebOnto: discussing, browsing, and editing ontologies on the Web, in: *proceedings of the 11th Knowledge Acquisition for Knowledge-Based Systems Workshop*, Banff, Canada (1998).
8. J. Tennison, and N.R. Shadbolt, APECKS: a tool to support living ontologies, in: *proceedings of 11th Knowledge Acquisition for Knowledge-Based Systems Workshop,* Banff, Canada (1998).
9. M. Arumugam, A. Sheth, and B. Arpinar, Peer-to-Peer Semantic Web: a distributed environment for sharing semantic knowledge on the web (2002); http://lsdis.cs.uga.edu/lib/download/ASA02-WWW02Workshop.pdf.
10. P. Mitra, M. Kersten, and G. Wiederhold, Graph-oriented model for articulation of ontology interdependencies, Stanford University Technical Note, CSL-TN-99-411, (1999) and in: *proceedings of the 7th Intl. Conf. on Extending Database Technology* (EDBT 2000).
11. E. Sunagawa, K. Kozaki, Y. Kitamura, and R. Mizoguchi, An environment for distributed ontology development based on dependency management, in: *proceedings of 2nd Intl. Semantic Web Conf.* (ISWC2003), edited by D. Fensel et al., LNCS 2870, Springer-Verlag Berlin Heidelberg, (2003), pp. 453–468.
12. Y. Sure, M. Erdmann, J. Angele, S. Staab, R. Studer, and D. Wenke, OntoEdit: collaborative ontology development for the Semantic Web, in: *proceedings of the 1st Intl. Semantic Web Conf.* (ISWC2002), Sardinia, Italy, (2002).
13. B.A. Farshchian, *A framework for supporting shared interaction in distributed product development projects*, PhD thesis, IDI-NTNU, Trondheim, Norway, (2001).
14. J. Estublier, Objects control for software configuration management, in: *Advanced Information Systems Engineering, proceedings of 13th Intl. Conf. CAiSE*2001*, edited by K.R. Dittrich, A. Geppert, and M.C. Norrie, Interlaken, Switzerland, (LNCS 2068, Springer-Verlag, 2001), pp. 359-373.
15. G. Wiederhold, An algebra for ontology composition, in: *proceedings of 1994 Monterey Workshop on Formal Methods*, (1994), pp. 56-62.
16. J. Hefflin, and J. Hendler, Dynamic ontologies on the Web, in: *proceedings of 17th National Conf. on Artificial Intelligence* (AAAI-2000).
17. H.H. Do, and E. Rahm, COMA - A system for flexible combination of schema matching approaches, in: *proceedings of 28th Intl. Conf. on Very Large Databases* (VLDB), Hong Kong, (2002).
18. A. Doan, J. Madhavan, P. Domingos, and A. Halvey, Learning to map between ontologies on the semantic web, in: *proceedings of WWW-02, 11th Intl. WWW Conf.*, Hawaii (2002).
19. W. Li, and C. Clifton, SEMINT: A tool for identifying attribute correspondences in heterogeneous databases using neural networks, *Data & Knowledge Engineering*, **33**(1) (2000), pp. 49-84.
20. J. Madhavan, P.A. Bernstein, and E. Rahm, Generic schema matching with Cupid, in: *proceedings of 27th Intl. Conf. on Very Large Databases* (VLDB), Roma, Italy, (2001), pp. 49-58.
21. S. Melnik, E. Rahm, and P.A. Bernstein, Rondo: a programming platform for generic model management, SIGMOD, (2003), pp. 193-204.
22. E. Mena, V. Kashyap, A. Sheth, and A. Illarramendi, Observer: an approach for query processing in global information systems based on interoperability between preexisting ontologies, in: *proceedings 1st Intl. Conf. on Cooperative Information Systems*. Brussels, (1996).
23. R.A. Pottinger, and P.A. Bernstein, Merging models based on given correspondences, in: *proceedings of the 29th VLDB Conference*, Berlin, Germany, (2003).
24. E. Rahm, and P.A. Bernstein, A survey of approaches to automatic schema matching, *The VLDB Journal* **10**(4), (2001), pp. 334-350.
25. X. Su, J.A. Gulla, Semantic enrichment for ontology mapping, in: proceedings of the 9th Intl. Conf. on Applications of Natural Language to Information Systems (NLDB'04), Manchester, UK, (Springer-Verlag, 2004).
26. A. Egyed, Compositional and relational reasoning during class abstraction, in: *proceedings of the 6th Intl. Conf. on the Unified Modeling Language* (UML), San Francisco, USA, (2003), pp. 121-137.
27. A. Egyed, and P. Kruchten, Rose/Architect: a tool to visualize architecture, in: *proceedings of the 32nd Hawaii Intl. Conf. on System Sciences* (HICSS), (1999).

F.6 Domain Knowledge-based Reconciliation of Model Fragments

Strasunskas, D., Lin, Y., and Hakkarainen, S. Domain knowledge-based reconciliation of model fragments. In A.G. Nilsson et al. (Eds.), *Advances in Information Systems Development: Bridging the Gap between Academia and Industry*[25], Springer, 2006, *in press*.

[25] Proceedings of the 14th International Conference on Information Systems Development (ISD 2005), Karlstad, Sweden, August 2005.

Domain Knowledge-Based Reconciliation of Model Fragments

Darijus Strasunskas[1], Yun Lin[2] and Sari Hakkarainen[3]

[1] Norwegian University of Science and Technology, Norway and Vilnius University, Lithuania. Darijus.Strasunskas@idi.ntnu.no
[2 3] Norwegian University of Science and Technology, Norway. (Yun.Lin, Sari.Hakkarainen)@idi.ntnu.no

Introduction

Modelling is the activity of formally describing some aspects of the physical and social world around us for the purposes of understanding and communication [5] that often is applied in the early phases of systems development: analysis and design. However, different people usually present different models even when given the same domain and the same problem. Same information about a system can be modelled on various levels of abstraction, from different viewpoints, and consider different aspects. Model heterogeneity generally arises due to the creative nature of the modelling activity. Other factors such as the richness of the modelling language [4], and the ambiguities of modelling grammars typically strengthen model heterogeneity.

Modelling is usually cooperative activity among several developers/ analysts, where a final model must be composed from different intermediate model fragments. The challenge in modelling is to arrive at a coherent, complete and consistent description of the problem in a particular domain. In this paper, we seek to answer the questions: “How can we relate different views and aspects in modelling?” and: “How can we manage the changes in a distributed modelling environment?”

Ontologies have been used in various roles for different types of consolidation purposes, e.g. [2, 3, 12] for data integration, or schema and ontology integration through upper level ontology. Similarly to conceptual models, ontologies are built with the aim of sharing knowledge, or definitions, with other people. In addition, they are created to support automatic reasoning. Here, we elaborate on how ontologies could be used as an intermediate medium for model consolidation. The focus is on end-user support for the individual developers. Their views and perceptions of the Uni-

Advances in Information Systems Development: Bridging the Gap between Academia and Industry. Edited by A.G. Nilsson et al., Springer, 2006

verse of Discourse (UoD) are integrated based on some explicit basic knowledge about the domain. Ontology is used as a reference point and built with the sole intention to share knowledge with others. We use ontology to define and formalize basic knowledge and the main objects in the domain. Below settings and an illustration of the problem are discussed, followed by a section presenting our approach in detail. Before concluding the paper, our approach is compared to the current state of the art.

The Complex Activity of Distributed Modelling

In general, modelling is a complex and difficult task. It is usually a creative and collaborative process, during which different stakeholders are expressing them at different levels of abstraction, focusing on different aspects, and producing different variants of each model. Thus, the problem here is how to support the management of logically and/or geographically distributed modelling tasks. The problem can be illustrated by the following scenario which will be used throughout the paper.

Consider a process of designing an offshore platform at an oil company. Different (groups of) engineers are responsible for modelling different parts and aspects of a new oil platform. One group is responsible for the pipeline system design, i.e. the technological equipment; another is modelling the platform on which all equipment will stand; and yet another is dealing with the capacity of oil extraction and production, i.e. the drilling and pumping devices.

Developers (groups) work separately having only weekly meetings to align and reconcile their models. During the meetings every developer goes through the changes and decisions made after the last meeting. Other developers know, based on previous experience and common knowledge, what impact those changes have on their own models. For instance, if the production engineer decides to increase the oil pumping production by a certain amount, the engineer responsible for pipe lines knows that some of the pipes should be changed to wider and thicker ones to support the increased pressure. Meanwhile, the engineer designing a platform can see an impact to her part of the work as the platform will need to carry heavier constructions built on it. The problem here is how to support this kind of collaboration activity by at least partially computerizing and automating this troublesome task of model reconciliation.

Such rough impact assessments are based on ad-hoc expertise shared by all developers engaged in this project. They are aware of the dependencies that hold between model elements, even if the parts are not explicitly or directly related. The dependency type considered here is the impact relationship where, if one element is modified, then the other is impacted by this modification, yet the elements are not otherwise physically or logically related. Other types of dependency are (but not restricted to) derived-from, composed-of, and based-on relationships.

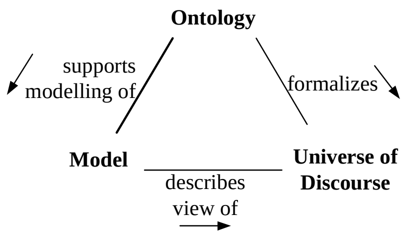


Fig. 1. Abstract view of our approach

Ontology as Intermediate Model

In order to relate the variety of model fragments we need to have common reference point. Our basic assumptions consider ontology, model and UoD as depicted in Fig.1. The approach is inspired by one of the linguistics' methods for describing the meaning of objects – the so called semiotic triangle [6].

UoD is basic knowledge about a particular domain. Ontology is used to represent (a portion of) UoD and to transform it into a man/machine understandable format. Ontology captures main concepts from a domain and represents relationships among the concepts in a machine readable and reasoning-able way. The goal is to capture common knowledge about which entities and what kind of dependency relationships they have in particular UoD. A model is instantiation of the ontology, where the basic theory supports representation of a particular problem. Consequently, our framework has two layers: an ontology layer for representing and formalizing a given UoD; and a model layer for modelling a problem (solution to the problem) within that UoD.

The advantage of this framework is that it separates the basic knowledge as the most reusable knowledge and places this in an ontology layer, keeping the layer of models separately. Ontology layer is composed of a set of concepts representing abstract entities in the real world, and relations among them that are normally based on the external and functional properties of the concepts. Model layer is instantiated abstract entities. To return to our scenario, each modelling of a variant of the oil platform is based on knowledge already captured in the ontology. Thus, ontology layer is used to reason about dependencies among modelled objects based on their properties and model layer is used for reconciliation of the situated model fragments. Relationships between the properties set a foundation for reasoning about the behaviour of objects in a domain.

Functional View

The model layer provides an environment where all model fragments are stored and managed. In order to achieve this there is one prerequisite for the distributed modelling activity and three iterative execution steps. Our approach consists of the following steps.

step 0 – Ontology development. This step is run only the first time when entering into new domain, when the knowledge about that domain is not yet formally described. The abstract objects, their properties and relationships are defined.

step 1 – Properties mapping. Mappings between function and external properties based on particular domain are produced.

step 2 – Collaborative modelling. During this step, models to solve the problem in question are developed, distributed and assessed. While modelling, the developers instantiate (decompose) the abstract concepts from the ontology layer.

step 3 – Model reconciliation. This step deals with model integration, change notification, i.e., the information captured in previous steps is used to reason about the dependencies among model fragments.

Model fragments are at different abstraction levels within the same domain. Therefore, certain concepts in the model fragments may be referred as being sub-class concepts of the concepts in ontology layer. These relationships, depicted as ‘kind_of’, and other formalized relationships among the concepts in the ontology layer are used to reason about the dependency between model fragments.

Functional and External Properties

In order to formalize the relationships, functional and external properties need to be defined for any ontology, in our scenario the oil drilling domain. A functional property of a thing is significant only when the function is used in a relationship with another thing. For example, the load limit of a platform needs to be mentioned only when the platform is expected to support things (constructions) put on it. Usually, external properties constrain the value of a thing’s functional property. An external property of a thing is present visibly even when the thing stands alone. The length, width, height and weight are external properties of a platform. Thus, a functional property is a property of an entity that denotes the main function of object in a particular UoD. An external property is a property of entity that denotes physically distinguishing features. Both properties are intrinsic properties in the sense of [13].

A contract holds between two things if they have a relationship, where some functional property of one thing and some external property of the other thing are mapped. Functional properties and external properties are mapped under certain conditions, which define a rule in an ontology as follows.

$$\text{Rule}: \ Func(x) \xrightarrow{map} Ext(y) \tag{1}$$

Recall the oil platform scenario. We demonstrate implementation of our approach in the next section. The external and functional[1] properties and the mappings are explained. We discuss a limited set of concepts, namely, `Platform`, `Oil`, `Pipe` and `PipeSystem`. The ontology is built in OWL using Protégé 3.0.

Ontology Building and Rules Definition

The ontology layer is used to capture the functional and external properties of entities in a UoD, and to represent the relationship between them. In the ontology fragment depicted in Fig. 2 in UML. `Platform` and `PipeSystem` are both defined as subclass of `Facility`. `PipeSystem` is composed of `Pipe` and `Oil`.

OWL distinguishes two types of properties, datatype property and object property. The former is an attribute of an object. The latter is a relationship between two objects. A datatype property can be regarded both as an external property and a functional property in our approach. For example, a `Platform` has a `support` relationship with `PipeSystem`, a functional `load` property with `Platform` and an external `weight` property of `Platform`.

An ontology model does not explicitly distinguish between external and functional properties. OWL is used to annotate them. A vocabulary **semAnn** is used to distinguish between them. The following is an OWL representation of the functional `load` property and the external `weight` property.

[1] Our definition of functional property is different from the OWL functional property.

```
    <owl:DatatypeProperty rdf:ID="load">
       <rdfs:domain rdf:resource="#Platform"/>
       <rdfs:range
   rdf:resource="http://www.w3.org/2001/XMLSchema#float"/>
       <SemAnn:functional_property rdf:ID="load"/>
    </owl:DatatypeProperty>

    <owl:DatatypeProperty rdf:ID="weight">
      <rdfs:domain>
        <owl:Class>
          <owl:unionOf rdf:parseType="Collection">
            <owl:Class rdf:about="#Pipe"/>
            <owl:Class rdf:about="#Facility"/>
          </owl:unionOf>
        </owl:Class>
      </rdfs:domain>
      <rdfs:range
rdf:resource="http://www.w3.org/2001/XMLSchema#float"/>
      <SemAnn:external_property rdf:ID="weight"/>
    </owl:DatatypeProperty>
```

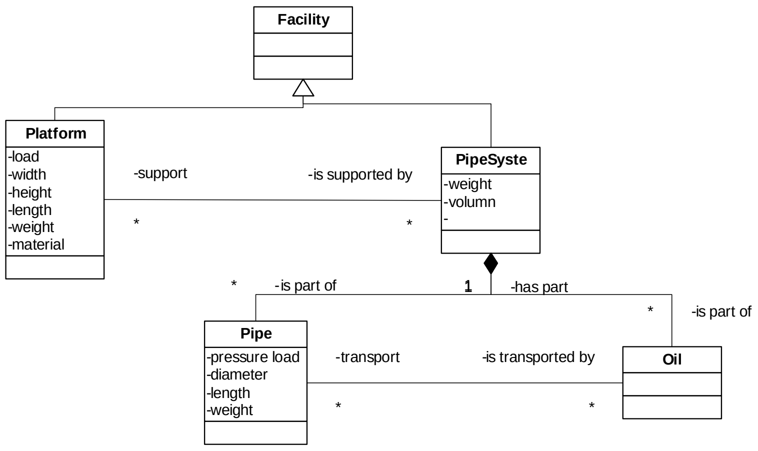


Fig. 2. Ontology fragment

In this domain ontology, we define the relationship between `Platform` and `PipeSystem` and assign them the related functional and external properties, followed by an example. Let *r* be a relationship (*r'* is a reverse relationship of *r*), *f* be a function, *EP* be a set of external properties, and *v* be value of a property. Then, fp^f is a functional property of a thing related to the function, ep^f is an external property of a thing constraining the function of another thing and ep_i is an external property. Finally, *ct* is a contract between two things and *cs* is a constraint of properties. Example:

```
r(Platform, PipeSystem) = 'support'
r'(PipeSystem, Platform) = 'is_supported_by'
```

```
f(Platform) = 'supporting'
EP(Platform)={epᵢ(Platform)|(i=1,2,…,v)}={Length, Width, Height,
                                           Material, Weight, …}
fp^supporting(Platform)=Platform.load
ep^supporting(PipeSystem)=PipeSystem.weight
fp^supporting(Platform)→ep^supporting(PipeSystem)=ct(Platform, PipeSystem)
v(Platform.load) <= v(PipeSystem.weight)
v(Platform.load) = cs(Platform.length, Platfrom.width, …)
```

Collaboration and Model Reconciliation

Model fragments in the model layer refer the knowledge that is stored in the ontology layer in order to 1) consolidate concepts, 2) find connection points, and 3) apply constraints when reconciling and integrating the distributed model fragments. Thus, a method for impact prediction and change propagation is needed.

The vocabulary **semAnn** is used to link the model instances to the abstract concepts that are defined on the ontology level. These concepts, including the relations, external and functional properties annotate the corresponding model fragments in local models as follows.

```
<semAnn:concept/relation/property
        rdf:resource="REFERENCE_ONTO:CONCEPT/
         OBJECT PROPERTY/DATATYPE PROPERTY" >
   MODEL FRAGMENT
</semAnn:concept/relation/property>
```

As two distributed models are connected, the relationships between models are built automatically. If one relationship in the model reflects a relationship in the ontology and a functional property is related to that relation, then there must be a functional property in one model and an external property in another. These two properties build an interface for the two models and they are expressed in constraints. If the functional property already is defined in the model and annotated as a property, then it is denoted a functional property. If the functional property is not defined in the model, it can be added referring to the functional property in the ontology. The related function and value of the functional property are annotated using `Rule:function` and `Rule:value`. The square brackets indicate optional statements in the annotation structure below.

```
<semAnn:property rdf:resource="REFERENCE_ONTO:DATATYPE
 PROPERTY">
   [MODEL FRAGMENT]
   <semAnn:functional_property/external_property>
   <Rule:function rdf:resource="RULE#FUNCTION"/>
   [<Rule:value>VALUE</Rule:value>]
   </semAnn:functional_property/external_property>
</semAnn:property>
```

Consider again the oil platform scenario in the OWL representation below. The `platformmodel` is built by an expert on platform engineering, and the `pipesystemmodel` is designed by another engineer. When two models are integrated to a `connectedmodel`, local concepts are aligned with common concepts.

```
   <semAnn:property rdf:resource="uri://domonto#length">
         <platformmodel:platform_length id="id2">
             …
            </platformmodel:platform_length>
     <semAnn:external_property>
        <Rule:value>580m</Rule:value>
        <Rule:cs rdf:resource="uri://rule#cs">
        </semAnn:external_property>
 </semAnn:property>

 <semAnn:property rdf:resource="uri://domonto#load">
         <platformmodel:carrying_capacity id="id6">
             …
            </platformmodel:carrying_capacity >
     <semAnn:functional_property>
        <Rule:function
          rdf:resource=" uri://rule#supporting"/>
        <Rule:value>500ton</Rule:value>
        <Rule:cs rdf:resource="uri://rule#CS">
     </semAnn:functional_property>
 </semAnn:property>

 <semAnn:property rdf:resource="uri://domonto#weight">
         <pipesystemmodel:weight id="id9">
   …
            </pipesystemmodel:weight>
     <semAnn:external_property>
        <Rule:function    rdf:resource="uri://rule#supporting"/>
        <Rule:value>200ton</Rule:value>
     </semAnn:external_property>
 </semAnn:property>

 <semAnn:relation rdf:resource="uri://domonto#support">
         <connectedmodel:hold id="cid5"/>
     <Rule:function    rdf:resource="uri://rule#supporting"/>
        <Rule:ct rdf:resource="uri://rule#CT">
 </semAnn:relation>
```

`Platform` can support `PipeSystem`; thus the properties of `PipeSystem` in the model should satisfy the limits of `load` of platform represented in another model. When an external property, e.g. `length` in the platform model is changed, the functional property `load` will be impacted according to the constraints defined in the rules. Since two models are connected, the contract between two models should be checked. Because the `weight` of `PipeSystem` is involved in the contract, the model fragments referring to the `PipeSystem` concept have to make corresponding changes. The changes can be traced using the annotation information in local models.

Two-Layered Approach Revisited – Semantic Reconciliation

In distributed models concepts as they are used may vary in a way they are denoted and represented. Those same-concepts-in-different-models need to be reconciled according to the ontology when models are to be integrated. Lets assume that context similarity has already been considered during the agreement and identification of the same concepts, as described in [9]. Here, distributed local models adjust their semantics referring to the ontology. The local models – `platformmodel` and `pipesystemmodel` – locate corresponding concepts in ontology. The relationships between the objects in ontology provide a clue for integrating the models. In the ontology of the scenario, OWL object `support` property is related to the functional `load` property and connects two objects – `Platform` and `PipeSystem`. That indicates how the `platformmodel` is to be integrated with the `pipesystemmodel`.

The rules, which contain the two functional properties, should be applied during the integration. These rules constrain the changes of models and are also used to check change impact on consolidated models. Three possible impacts are: 1) changes on the functional property may impact other models (e.g., changes on the range of `load` of `Platform` will impact the maximum `weight` of `PipeSystem`), 2) changes on external property of one object may impact its functional property (e.g., changes on `length` of `Platform` will impact its `load`), and 3) changes on external properties may impact other models (e.g., changes on `length` of `Platform` will impact maximum `weight` of `PipeSystem`). The procedure of checking for change of property is as follows.

```
if one property in a local model is changed
{ check the property in ontology level;
  if the property is functional property
      if the functional property is related with
          Object_Property
         { if the functional property is involved in con-
            tract-rules
                check changes on the other object which is
                involved in the contract-rules;
          else
                check changes on the other object which is
                involved in this Object_property;}
  else
       if the property is involved in constraint-rules
         { check changes impacting other properties involved
            in the constraint-rules;}
   }
```

Our preliminary prototype is implemented in Python. The main interface window consists of 5 panels: 1) a panel listing of model fragments stored in a repository; 2) a modelling panel for editing instantiated (related or associated) abstract entities; 3) a panel for ontology browsing; 4) a notification panel for listing changes and their impacts; and in addition 5) a chat panel for discussion between team members.

Related Work

Ontology is commonly defined being an explicit (formal) specification of a conceptualization [1] in the recent literature. Therefore, application of ontologies is in resolving semantic heterogeneity. With respect to the integration of data sources, ontologies can be used for the identification and association of semantically corresponding information concepts [12]. Ontologies are previously used to provide semantic interoperability in information sharing e.g., [2, 3, 12]. The semantics of a resource in a particular domain can be explicitly defined by associating concepts, terms and various information resources with concepts in an ontology.

Further, ontologies can be used as means to abstract from different representation formats and to relate various product fragments at different abstraction levels. Ontologies (domain models) are used in [8] to relate various fragments of system specification to establish dependency relationships for change impact prediction. The end product of system development is seen as a collection of loosely coupled product (specification) fragments from various perspectives that focus on different aspects. The co-ordination of the development process and integration of different product fragments utilizes a common reference layer, i.e. ontology.

An on-going research project [7] is looking at supporting requirements elicitation and composing software from re-usable architectures, frameworks, components and software packages. The use of ontology and its reasoning mechanism helps to maintain semantic consistency. Ontology system there has two layers; one for requirements elicitation and the other for re-usable parts. The ontology system bridges gaps between a requirements specification and an architectural design at a semantic level by establishing relationships between the two layers [7].

An interesting approach is described in [14], where knowledge is organized in knowledge grid. They separate between epistemology and ontology treating both as inseparable profiles of the unified human cognition process. The epistemology mechanism used as a semantic description tool to reflect human subjective cognition. The mechanism helps humans understand and relate their knowledge to the one captured in ontology. Ontology reflects people’s consensus on semantics [14].

The work that has been done so far in the area of development and maintenance of ontologies mainly has focused on one ontology, which is edited by the developer. On another hand, there are some tools which allow collaborative ontology creation. For instance, Hozo [10] environment for distributed ontology development is based on splitting ontology into component ontology and establishing dependency between them. The target ontology is obtained by compiling the component ontologies, based on predefined links between them.

In summary, there are different application areas for ontology usage. We find our approach novel as ontologies are used as supervising guidelines during modelling activity. The approach allows checking models under development whether they are semantically correct within particular domain, i.e., how a model corresponds to basic domain knowledge captured in ontology.

Concluding Remarks and Future Works

A vision of a methodological approach to facilitate management of collaborative logically or geographically distributed modelling activities is presented. The approach is based on distinguishing two main layers: an ontology layer; and a model layer. Ontologies are used as a medium for common knowledge representation and as a guide for models reconciliation. The ontology layer contains a set of predefined valid relationships for the creation of situated models. Further, it provides reasoning about relationships between model fragments. We capture two types of object properties in ontology – functional and external property. Relationship between those properties is the foundation for reasoning about the behaviour of objects in a particular domain, e.g., how change of one property influences the change of another. We provide the motivation for our research, discuss the settings and provide conceptual description of the approach followed by scenario that illustrates the applicability of our approach.

There are some remaining challenges to our approach and to the current version of our prototype, however. One is to create an algorithm for automatic update of the models based on both observed changes in the model fragments and on formalized relationships in the ontological layer. Further, description of the rules in a related web-based syntax would be an advantage for the approach as it will allow usage of the same reasoning mechanism as in the ontology layer. The proposal for *Semantic Web Rule Language* (SWRL) [11], whose syntax is based on a combination of OWL DL and the Datalog sublanguage of RuleML, is a good candidate for the further implementation.

References

[1] Gruber TR (1993) A Translation Approach to Portable Ontology Specifications. Knowledge Acquisition, vol 5, no 2, pp 199-220
[2] Gruber TR (1991) The Role of Common Ontology in Achieving Sharable, Reusable Knowledge Bases. In: Allen JA, Fikes R, Sandewall E (eds) Principles of Knowledge Representation and Reasoning. Morgan Kaufman
[3] Kashyap V, Sheth A (1994) Semantics-Based Information Brokering. Proc of the 3rd Intl. Conf on Information and Knowledge Management
[4] Moriarty T (2000) The Importance of Names. The Data Administration Newsletter 15
[5] Mylopoulos J (1992) Conceptual Modeling and Telos. In: Loucopoulos, Zicari (eds) Conceptual Modeling, Databases, and CASE. Wiley
[6] Ogden CK, Richards IA (1923) The Meaning of Meaning. 8th ed. Harcourt, Brace & World Inc. New York
[7] Saeki M (2004) Ontology-Based Software Development Techniques. ERCIM News, no 58, p 14
[8] Strasunskas D, Hakkarainen S (2003) Process of Product Fragments Management in Distributed Development. Proc of the (CoopIS'2003), Springer, LNCS2888
[9] Strasunskas D, Lin Y (2004) Model and Knowledge Management in Distributed Development: Agreement Based Approach. Proc of the 13th Intl. Conf. on Information Systems Development (ISD'2004) Vilnius Lithuania
[10] Sunagawa E, Kozaki K, Kitamura Y, Mizoguchi R (2003) An Environment for Distributed Ontology Development Based on Dependency Management. In: Fensel D et al (eds) LNCS 2870. Springer
[11] Horrocks I, Patel-Schneider PF, Boley H, Tabet S, Grosof B, Dean M (2004) SWRL: A Semantic Web Rule Language Combining OWL and RuleML
[12] Wache H, Vogele T, Visser U, Stuckenschmidt H, Schuster G, Neumann H, Hubner S (2001) Ontology-Based Integration of Information – A Survey of Existing Approaches. In: Stuckenschmidt H (ed) IJCAI-01 Workshop: Ontologies and Information Sharing
[13] Wand Y, Weber R (1995) On the Deep Structure of Information Systems. Information Systems Journal, vol 5, pp 203-223
[14] Zhuge H (2004) China's e-Science Knowledge Grid Environment. IEEE Intelligent Systems, vol 19, no 1, pp 13-17

Index