Abstract

Systems that efficiently provide support for adaptive work processes in a mobile environment do not exist according to our knowledge. Such systems would increase efficiency and safety in environments where work is inherently mobile, ad-hoc, and requires input from a set of heterogeneous sources. Traditional work support systems are normally not capable of dynamic change and plans must be made before work is started. This conflicts with most work processes, which are dynamic and where plans cannot be completely pre-defined.

Current workflow systems are for the most part not capable of handling dynamic change in the workflow process execution. Those that do exist are geared more towards long-term adaptability of the workflow process and not towards in-situ planning of activities.

In this report we provide an overview over current research related to adaptive workflow, activity theory, situated actions, and context-awareness. Then, we further explore the concept of adaptive workflow and context-awareness and how this can be implemented in a prototype workflow enactment service. A set of requirements for such a system is elicited from this exploration. We also provide a possible scenario for usage of adaptive context-aware workflow technology.

From these requirements we have created an overall architecture that supports adaptive context-aware mobile work. Our focus within this architecture is on context-aware adaptive workflow systems.

We finally present the design and implementation of a prototype application supporting context-aware adaptive mobile work processes. This prototype has been named PocketFlow and is implemented in embedded visual C++ for Microsoft PocketPC 2003 second edition.

Preface

This report is a result of work on the master thesis by Frode Hauso and Øivind Røed at the department for Computer and Information Science (IDI) at the Norwegian University of Science and Technology (NTNU) in the spring of 2005.

We would like to thank our supervisor, PhD Fellow Carl-Fredrik Sørensen, for his support and invaluable insights, ideas, and comments during our work. In addition, we would like to thank Morten Aarbakken and Aker Verdal for help on providing a scenario for our work, and Thirunavukkarasu Sivaharan for providing the CORTEX Publish-Subscribe Component Framework.

Trondheim 15th of June 2005

Frode Hauso Øivind Røed

Contents

List of Figures

List of Tables

Part I

Introduction

Chapter 1

Introduction

This chapter presents the background, motivation, and problem definition for our work and outlines the structure of this report.

1.1 Background

Our master thesis is related to the MOWAHS research project [34] at NTNU [38]. MOWAHS is a project carried out jointly by the Software Engineering and Database Technology groups at IDI [25] and has two main parts; process support for mobile users using mobile devices, and support for transactions/workspaces incorporating work documents.

The MOWAHS project states three main goals for their research:

- Help understanding, and continuously assess and improve work processes in virtual organisations.
- Provide a flexible, common work environment to execute and share real work processes and their artifacts, applicable on a variety of electronic devices.
- Distribute the results to the community at large.

The approach taken by MOWAHS to achieve these goals is to iteratively define a flexible work environment for virtual organisations. This work environment should support processes and their transactions. Such a support is often implemented in test beds for process support that realise real scenarios.

This report partly addresses all three goals by proposing a work environment on mobile devices, which is tested by a prototype implementation of an adaptive, ad-hoc workflow system. Results from this work can be used to help the understanding of adaptive, mobile work processes, and thus contribute to the overall goals of MOWAHS.

1.2 Motivation

The motivations for developing support for mobile, adaptive work processes in a context-aware workflow system are described below.

Increased efficiency and safety in workflow systems

Adaptive, mobile work processes can provide great benefits in industrial applications. Better support for process enactment in mobile environments can be achieved by taking the mobile work environment into account when performing work, and thus incorporate and integrate a more dynamic behaviour into the enactment. This could possibly increase efficiency because the system can automate the collection of data and the execution of services. Safety can also be increased since workers will have an increased awareness of the environment. Also the knowledge of people and actuators in an environment can help reduce safety risks, where an environment may include applications that preserve a safe work environment based on sensed data and embedded knowledge about the environment.

Support for mobility in computer systems

The evolution of new technology has in the recent years resulted in an increased use of devices such as PDAs and mobile phones. This has led to lower prices and more applications available for the consumer marked. Users of such devices can benefit from systems that take contextual information into account, and provide relevant information and services to the user based on guidelines, preferences, activities, etc.

The planning paradox

Traditional workflow systems have difficulties supporting unexpected changes and deviations from the process model. Organisations need to organise their work according to plans even though work is by nature ad-hoc and situated. This is referred to as the planning paradox [5], and is a motivating factor for creating support for work processes which is adaptive and reactive to changes in the environment.

1.3 Problem Definition

We can envision a scenario in a offshore shipyard (Chapter 8) where a workflow (Chapter 3) is highly dynamic and can not be properly defined until the worker is in a specific location. In such an environment there is a need to adapt the workflow process at run-time which may include the use of context-sources (Chapter 4) to extract required information from, and sending actuation orders to, the physical environment while executing the workflow. There may also be several mobile workers in the same location, often with conflicting interests. Since the workflow can not completely be specified before the worker is at the correct location, there is a need for some sort of template system (Chapter 3.6.3) where the template contains the overall high-level workflow. This template is then instantiated by using information retrieved from the context sources in the worker's environment.

The rapid improvement in sensor technology, mobile equipment, and wireless communication enables us in a not too distant future to build workflow systems that support a higher degree of adaptivity and context-awareness. To our knowledge there has not yet been built such a workflow system. The main objective for this thesis is to develop a

workflow prototype which uses contextual information from the environment to adapt a workflow process to it's environment. By creating an architecture (Part V) and a proof of concept prototype (Part VI) we are able to test our thoughts on these issues with our specific implementation. We will in the rest of the report use PocketFlow as the name of our prototype.

Chapter 2.1 describes in more detail the problems we set out to solve.

1.4 Limitation of Scope

Issues related to wireless communication will not be dealt with in this report, and we assume reliable and stable connectivity. We also assume that the client will be disconnected from the central workflow enactment service.

Even if such issues still poses important challenges, they are not addressed in our work but are thoroughly examined in other research, e.g. [42] which deals with challenges in mobility management and wireless communication.

1.5 Outline of the Report

Part I - Introduction

- Chapter 1 - Introduction: Contains background, motivation, and problem definition.
- Chapter 2 - Research Methods: Presents the research questions examined in this thesis as well as the research methods used to provide the results.

Part II - State-of-the-Art

- Chapter 3 - Workflow: Gives an introduction to workflow technology with a strong emphasis on adaptive workflow processes.
- Chapter 4 - Context-Aware Computer Systems: Gives an introduction to context and context in computer systems.
- Chapter 5 - Activity Theory and Situated Actions: Gives an introduction to activity theory and situated actions and discusses the difference between the two theories.
- Chapter 6 - Sensors and Actuators: Gives an introduction to the notion of smart dust, wireless sensors and actuators, and the intelligent artifact paradigm.
- Chapter 7 - Development Frameworks: Presents the frameworks used in the development of PocketFlow.

Part III - Scenario

- Chapter 8 - Application Scenario - Aker Verdal: Presents a possible scenario for usage of the PocketFlow prototype. The current work processes are described, and then new work processes that takes benefit of context-awareness are introduced.

Part IV - Requirements

- Chapter 9 - Requirements for the PocketFlow Prototype: Presents functional and non-functional requirements for the PocketFlow prototype, as well as COTS components and the technical constraints for the architecture.
- Chapter 10 - Use Cases: Presents the use cases that form the basis for the architecture and design of PocketFlow.

Part V - Architecture and Design

- Chapter 11 - The PocketFlow Architecture: Presents the high level architecture for PocketFlow.
- Chapter 12 - Design: Presents the implementation of the architecture from Chapter 13.

Part VI - Implementation and Testing

- Chapter 13 - Implementation: Gives a discussion of implementation metrics and presents the project structure as well as code examples.
- Chapter 14 - Testing: Presents the unit tests for PocketFlow as well as a description of the debugging process.

Part VII - Discussion and Conclusion

- Chapter 15 - Discussion: Presents a discussion of the results from this thesis.
- Chapter 16 - Further Work: Presents what parts of the PocketFlow prototype that need further work.
- Chapter 17 - Conclusion: Gives the concluding remarks for this thesis.

Part VIII - Appendix

- Appendix A - Tools: Presents the tools used when developing the PocketFlow prototype.
- Appendix B - Class Diagrams: Presents the UML diagrams for the PocketFlow prototype.
- Appendix C - Workflow Fragments XML Schema: Presents the XML schema for workflow fragments.
- Appendix D - Actual Parameters XML Schema: Presents the XML schema for actual parameters used in the workflow enactment.

- Appendix E - XPDL Template Example: Presents the XPDL workflow template used in Appendix H.
- Appendix F - Workflow Fragments File Example: Presents workflow fragments file used in the XPDL in Appendix H.
- Appendix G - CLIPS Knowledge Base: Presents the CLIPS knowledge base used in the example workflow.
- Appendix H - Context-Aware Workflow Example: Presents a context-aware workflow example included the log file from the execution.
- Appendix I - CD-ROM: Gives a presentation of the accompanying CD-ROM.
- Appendix J - Glossary: Gives a short description of central terms in this thesis.

1.6 Readers Guide

This report contains both conceptual descriptions as well as technical descriptions of our work on adaptive mobile work processes. The state-of-the-art part (Part II) provides the reader with discussions on relevant topics and is especially useful for readers new to this research area. Own contribution is documented in the scenario (Chapter 8), requirements (Part IV), and architecture (Chapter 11) in a conceptual manner. For an additional more detailed, technical description, the reader is referred to the design (Chapter 12), implementation (Chapter 13), and testing (Chapter 14) chapters. Discussion and conclusions (Part VII) contain a summary of our findings and is recommended for all readers.

Chapter 2

Research Methods

This chapter presents research questions, research methods and software engineering methodologies used in the work on our master thesis.

2.1 Research Questions

We have defined important aspects of our work by stating a set of research questions which are listed below.

RQ1 How can a workflow process adapt to environmental changes, and how can a workflow process modify the environment?

This research question relates to how a workflow client can use contextual information from the environment to evaluate transitions for work processes and possibly open new paths to process goals. A workflow client may perform scheduling of activities, synchronisation of environmental context with related activities, or perform actuations as a reaction to contextual changes. Such a process enactment with correct state transitions should be achieved also when the workflow client is disconnected from a central workflow server.

RQ2 How can we achieve automatic definition of activities based on process goals and environmental context?

Template work processes can be instantiated by using context which may make it possible to achieve automatic definition of activities based on contextual conditions and process goals. This kind of adaptivity for work processes is a combination of using pre-planned templates and contextual information captured in-situ.

RQ3 How can we achieve ad-hoc workflow enactment in a mobile computing environment?

For instance how can we decide when a workflow construct needs to be created without a priori knowledge of the environment around the mobile worker?

The rest of this chapter covers the research methods we used when working with these research questions.

2.2 Engineering Approach

Zelkowitz [64] defines the engineering approach.

> Engineers develop and test a solution to a hypothesis. Based on the results of the test, they improve the solution until it requires no further improvement.

An engineering approach is suitable for our work since the main goal for our master thesis is to develop a solution to the problem definition stated in Chapter 1.3. We have developed a proof of concept prototype which is an improvement to previous developed prototypes by Nguyen and Nødtvedt [39].

2.3 Literature Survey

To answer our research questions, we have done a literature survey. This survey gives us an understanding of the relevant aspects of our problem domain. The Internet provided us with articles, journals, etc. from research sites where NTNU have licence.

- Association for Computing Machinery (ACM) [1]
- Springerlink [50]
- Cite Seer [49]
- IEEE Xplore [26]

In addition, a large number of publications have been made available through a shared bibliography in the MOWAHS project.

2.4 Scenario Building and Requirement Elicitation

This report includes one scenario which provides a showcase for a possible deployment environment for our proof-of-concept prototype. This scenario also helps us to understand relevant requirements for systems with support for adaptive, mobile work processes.

Scenario Building - Field Study at Aker Verdal

The scenario is based on a limited field study of a yard environment by visiting Aker Verdal in Nord-Trøndelag. During this visit we were shown their production facilities which provided us with valuable information. Our application scenario in Chapter 8 is based on this field study. We have previously elaborated other scenarios and analysed related requirements in a previous project [21].

Requirements Elicitation

We are evolving previously stated requirements ([21] and [39]) by incrementally developing a proof-of-concept prototype. Requirements are found and refined with the increased understanding of the problem domain.

2.5 The XP Development Methodology

We need a software engineering methodology that takes into consideration that our master thesis only spans five months, and that requirements most likely will change as a result of our increased understanding of the problem domain. In order to rapidly build a running prototype, we need a methodology which facilitates implementation and testing of small increments in an evolutionary manner. A software engineering methodology which is known to work for small teams and speed up development is eXtreme Programming (XP) [19].

Extreme programming is a lightweight discipline for software engineering which is based on principles of simplicity, communication, feedback, and courage. It can be summed up in a set of practises which aim to help developers achieve rapid development of software in a environment of changing requirements. Important XP practises are [19]:

Coding standards The use of coding standards makes communication easier between both current developers and future maintainers.

Simple design XP is concerned with making the simplest design possible and not spend time on features that might not be needed later. This helps keeping the cost of change low and makes the project more flexible to changing requirements.

Small releases XP stresses that releases should be as small as possible while delivering enough value to make them worthwhile.

Testing In XP the focus is on validating the software at all times. Programmers write tests cases before they write code, often called test-driven development. This way of doing test first design may help us to keep focus on what we are trying to achieve and develop software that fulfils our requirements.

Refactoring Refactoring is a technique which involves changing existing code without changing the functionality. This is a good way to improve design and code to reflect our increased understanding of the problem domain.

We will only use those parts of the XP metodolgy that we find relevant for our work. These practises have helped us reach a satisfactory implementation of our previously proposed architecture in [21].

Part II

State-of-the-Art

Chapter 3

Workflow

Since the architecture presented in this report heavily depends on workflow technology, we will in this chapter give a short introduction to workflow and workflow management systems before we present some research and solutions to adaptive workflow and workflow templates. While this topic is covered in our previous report [21], we repeat the most important parts.

3.1 Benefits of Workflow Technology

Workflow technology is created to support automation of business processes where documents, information, or tasks are passed between participants according to a defined set of rules to achieve, or contribute to, an overall business goal [23]. The goal of our architecture is a system that can support mobile work processes. The scenario in Chapter 8 show that an implementation of such a system requires automation of procedures like sending information about activities and sending new activities based on a set of rules. Workflow technology will therefore give us the required theory and model to more easily create such a system. In addition, we have the benefit from an industry standard which should increase the life-span of our implementation. While workflow technology is very useful, it is also a very large field with a large number of standards. We will therefore take care to only use those parts that give us the most benefit.

3.2 Workflow Example

Figure 3.1 shows a simple workflow with a set of activities and the transitions between them. A workflow process can be considered to be a directed graph of activities. After an activity has been split, we dub that part of the graph as a *thread of execution*. For instance, when *Activity2* split into three new activities, each of the activities become separate threads of execution. When *Activity3* joins with *Activity6* the *Activity3* thread of execution is terminated. The same occurs when *Activity5* joins with *Activity7*.

3.3 Terminology

To provide a common ground for further discussion we will look at some of the workflow terminology. Workflow is standardised by the Workflow Management Coalition (WfMC)

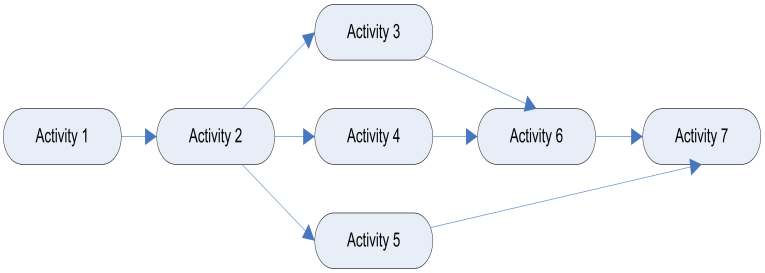

Figure 3.1: Workflow example

[56], a non-profit, international organisation that promotes the use of workflow by establishing standards and terminology for connectivity between workflow products. The coalition is the primary standards body for workflow. The workflow terminology is found in [11] and the most important terms are presented in Appendix J.

Figure 3.2 shows the WfMC definitions [11] and the relationships between them.

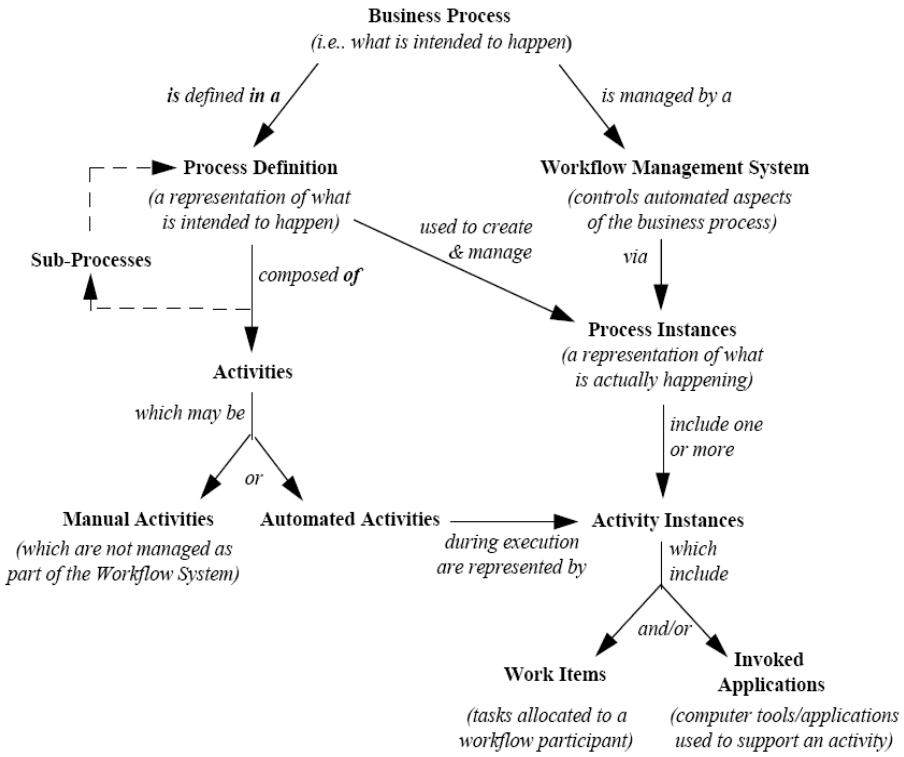

Figure 3.2: Workflow definitions

3.4 Types of Workflow Systems

There exist many types of workflow systems that support a wide range of work environments. We will for our work create a workflow system that is both collaborative and ad-hoc. A short introduction of these system types follows.

3.4.1 Collaborative

A collaborative workflow focuses on groups working together toward common goals, also called groupware. We have in our architecture taken collaboration into account, but it is not the main focus of our work.

3.4.2 Ad-Hoc

Ad-Hoc workflows are created on-the-fly by the current participants, and the process definitions are very flexible. We will have a more detailed look at ad-hoc workflow systems in Chapter 3.6. Our work will focus on single workers or small groups of workers in an ad-hoc environment. In essence our solution is an ad-hoc workflow.

In the next section we will explore the WfMC's workflow reference model.

3.5 WfMC's Workflow Reference Model

To support interaction between workflow products, WfMC has defined the workflow reference model shown in Figure 3.3. This model is divided into five interfaces. Interface 6-8 are still under development and are currently not a part of the official workflow reference model. In our work, we try as much as possible to adhere to the standards described in interface 1, but since the workflow reference model is not created for dynamic ad-hoc workflow systems, there are some deviations. Interface 2 could potentially be a benefit for someone wanting to use our workflow engine in their own work, but because of time constraints we will only use this specification when it does not interrupt our work. A short description is included anyway.

The reference model provides a common vocabulary for describing business processes, a functional description of the software components needed to support workflow, and the definition of interfaces between the software components. It tries to be as technology neutral as possible.

3.5.1 Interface 1: Process Definition

This interface integrates the process definitions to the workflow enactment service in the form of an interchange Process Definition Language (PDL) and API [1] calls.

The process definition provides an environment for a rich description of a process that can be used for the following [55]:

- Act as a template for the creation and control of instances of that process during process enactment.

[1] Application Programming Interface

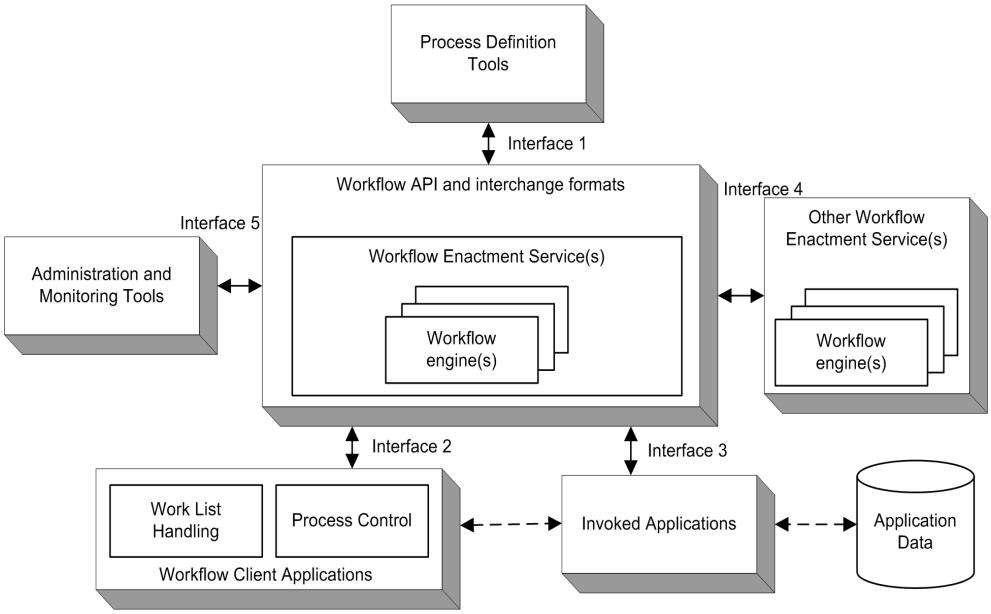

Figure 3.3: WfMC's workflow reference model

- For simulation and forecasting.
- As a basis to monitor and analyse enacted processes.
- For documentation, visualisation, and knowledge management.

Using a standard method of representing workflow process definitions makes it possible to use standard tools for modeling business processes, while reducing our workload by not having to create our own process definition language.

Meta-Model

To provide a common method to access and describe workflow definitions, a workflow process definition meta-model has been established. This meta-model identifies common entities within a process definition. A variety of attributes describe the characteristics of this limited set of entities. Based on this model, vendor specific tools can transfer models via a common exchange format we will describe next. Figure 3.4 shows the meta-model promoted by the WfMC. The *System and Environmental Data* element is of special interest to us. This element contains data which are maintained by the workflow management system or the local system environment that can be accessed in the workflow process enactment, and used in the evaluation of conditional expressions. In our work, we will use this element as an interface to context sources (Chapter 4.1).

Process Definition Language: XPDL

To support a variety of different tools that analyse, model, describe, and document business processes, there is a need for a standard interchange format that transfer workflow process

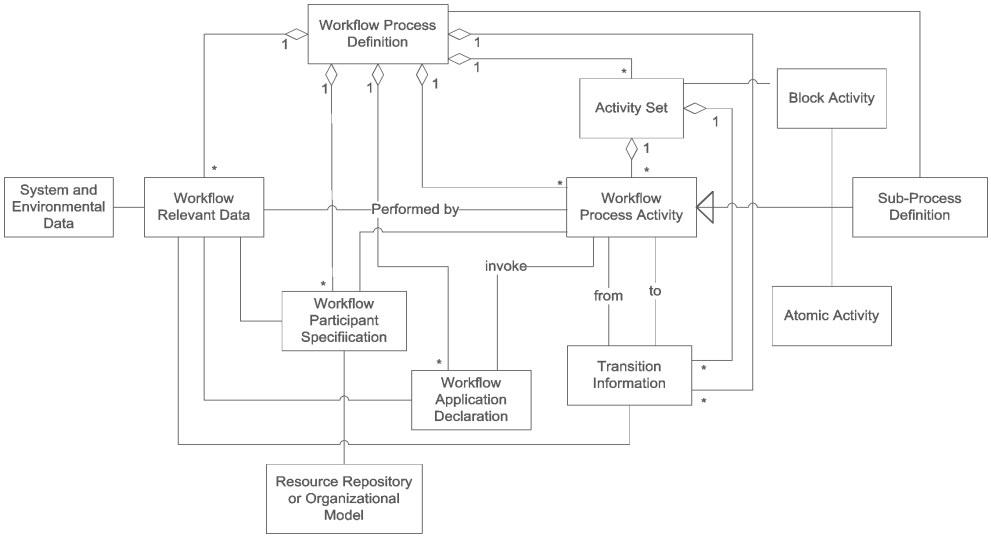

Figure 3.4: WfMC's workflow reference meta-model

definitions between the separate products. WfMC has created and promotes a formal specification, the XML [62] Process Definition Language (XPDL) [55]. XPDL supports all the entities in the workflow reference meta-model. See [55] for definitions of all the entities and their corresponding XPDL tags, and Appendix E for an example.

We will rely on XPDL for loading of workflow processes. In addition we will use XPDL to store workflow templates and fragments (discussed in Chapter 3.6.3).

3.5.2 Interface 2 and 3: Workflow APIs

Interface 2 and interface 3 have been combined by the WfMC, and cover the workflow APIs (WAPIs). These APIs allow external applications to access management engine services. This makes it possible to have a single end-user interface and function set to easily replace the workflow engine. WAPIs have traditionally been written for the C language, but have in the recent years also been written for the Java language.

This set of APIs is not that important to us since we are not making a commercial workflow system that needs to interoperate directly with other workflow systems. That type of communication is in our system maintained by using context.

3.6 Adaptive Workflow and Exception Handling

In this section, we present exceptions and exception handling in workflow systems as well as research on adaptive workflow processes.

3.6.1 Exceptions

Kammer et al. [28] divide exceptions into two main categories: expected exceptions and unexpected exceptions. Expected exceptions are exceptions that are modelled into the

workflow at design time, while unexpected exceptions naturally can be very hard to model at design time. Expected exceptions are not very interesting in our work, since they are already covered by the WfMC workflow reference model. We will therefore only look at unexpected exceptions and possible methods of handling them without manual intervention. It is also important to note that we only discuss exceptions in the business process execution. Other exceptions like network disconnections, varying network bandwidth, and hardware or software malfunctions are not covered here.

Exceptions occur at three different levels in the organisation [28]:

- **Employee level** Impact is limited to a single individual.
- **Group level** Impact stretches across a group working on the same project, process etc.
- **Organisational level** Impact is organisation wide, covering more than one group.

Unexpected exceptions can be classified into three groups. The first group is minor exceptions that can safely be ignored, which Kammer et al. [28] calls *noise*. Another group of exceptions contains those that are relatively unique to a workflow process, but still force the workflow process to change. They call this type of exceptions *idiosyncratic*. The last group of exceptions is *evolutionary* exceptions that require changes in the organisational workflow model. In our work, we try to reduce the number of unexpected exceptions by providing workflow fragments and templates to automatically generate new activities.

Since the focus of our report is adaptive mobile work for single workers and small groups of workers, we will not discuss exceptions and adaptations at the organisational level, but instead concentrate on *idiosyncratic* exceptions and *noise*. See Table 3.1 for the relationship between the types of exceptions and what changes they inflict on the workflow model.

Exception Type	**Change in Workflow Model**
Noise	No impact.
Idiosyncratic exceptions	Changes to specific instance of workflow, but the workflow type (the general model) remains the same.
Evolutionary exceptions	Evolution of general workflow model, affecting future instances of work process as well.

Table 3.1: Relationship between types of workflow exceptions and changes to the workflow model

3.6.2 Handling Exceptions

If we want to avoid unexpected exceptions stopping our workflow execution, we need methods to automatically detect and avoid, ignore, or handle the exceptions. Kammer et al. [28] suggest that the system should tolerate minor deviations, handle changes to the process instance, and support evolution and optimisation of the process model. We will only cover the two first suggestions since the third handles *evolutionary* exceptions.

Tolerating Minor Deviations

Minor or insignificant deviations to the workflow process, or noise, can safely be ignored and a workflow system should be able to safely detect and ignore them. This may for instance be solved by setting limits on how much fluctuation of the operational parameters that can be accepted.

Handling Changes to the Process Instance

Exceptions with larger impact on the workflow process than noise traditionally involve a stop in the process execution, and manual interaction by one or more workflow participants before the workflow process can be started again. This can be avoided by letting the workflow process instance be adaptive or ad-hoc. We will in the next section describe what an adaptive or ad-hoc workflow process is.

3.6.3 Adaptive Workflow

Cardoso et al. [8] classify the different types of dynamic change into two main categories: *primitive change* and *composite change*. Primitive changes are composed of atomic changes that can either be fully executed, or not executed at all. They further divide primitive changes into *immediate changes* and *incremental changes*. Immediate changes can be introduced without changing the correctness and consistency of the workflow, while with incremental changes, the entire change cannot be executed without introducing inconsistency. Composite changes are sequences of primitive changes.

Atluri and Chun [4] define three types of dynamic changes:

- **Profile Change** Changes to the goals of the business process.
- **Exceptions** Unexpected exceptions that arise during a task execution.
- **Rule Change** Changes to business rules and regulations.

If we consider exceptions to be a type of change [8], an adaptive workflow is a workflow able to adapt to exceptions and external changes and input. External changes and input can be profile change, rule change, or context sources (see Chapter 4). An adaptation can result in restructuring the workflow, either by inserting, deleting, modifying, redoing, or undoing activities.

Methods for supporting adaptive workflow include:

Late Binding

Defer binding of objects to the workflow process instance to the last moment. This results in a separation of an object's data from its behaviour. We will in Chapter 12.1.2 discuss a design pattern that allows us to achieve this.

On-The-Fly Workflow Process Composition

The definition of the workflow process is not completed until run-time when it is constructed from a workflow template. The template is instantiated at run-time by connecting to external actors, sensors, and actuators needed by the workflow process.

Partial Execution

Allow the execution of incomplete workflow processes, or workflow process fragments. By not requiring the workflow process to be complete, we can for instance execute all activities that are complete while waiting for later activities to be completed. For instance, when waiting for a sensor to come into the range of the workflow enactment system.

Reusable Process Fragments and Component Libraries

Fragments, stubs, or templates for work processes or tasks are stored in a data store either on the workflow client, distributed in the environment, or on a centralised server.

Creating Workflow Processes On-The-Fly

Mangan and Sadiq [31] introduce an approach to building flexible workflows. We have taken their model and converted it into a model more closely resembling the WfMC reference model. The model consists of

- a set of fragments from which an instance can be built. These fragments are stored in a fragment pool (data store). From these fragments we can build
  - a set of activities.
  - a complex sub-process.
- a set modeling constructs that can be used to compose the fragments for a given instance.
  - Sequence.
  - Fork / Synchronise (AND Split / AND Join in workflow terminology).
  - Multiple executions.

  Mangan and Sadiq [31] propose to not include choice-merge construct to reduce complexity and removing the possibility of deadlocks.
- a set of constraints that will define the rules under which valid instances can be built. Mangan and Sadiq [31] identify three levels of constraint specification.
  - Selection constraints.
  - Termination constraints.
  - Build constraints.

Selection Constraints Selection constraints determine which workflow fragments should be included when building the workflow instance.

Mangan and Sadiq [31] divide selection constraints into three types.

- Prerequisite constraints which indicate that the fragment is dependent on e.g. a completed activity.
- Companion constraints which are another type of dependency where the fragment requires other fragments which must also be selected.
- Incompatible constraints. The fragment may be incompatible with other fragments.

Termination Constraints Rules determining when the goal of the workflow process has been obtained and the workflow process needs to be terminated.

Build Constraints This includes rules that may for instance specify availability of resources, maximum load, and performance requirements.

Mangan and Sadiq [32] present a method for mapping constraints to a relational data base, including an ORM[2] specification. They propose to use SQL[3] to check for constraint violations.

We believe the approach for building flexible workflows proposed by Mangan and Sadiq [32] is a good solution for our problem of enacting adaptive workflow processes, but we further believe a system supporting this approach will be too complex for our application. We will therefore use some of their ideas to create a simpler, less complex system while still making sure it can easily be expanded to a more complete system in the future.

Workflow Templates

Atluri and Chun [4] introduce the notion of self-describing workflows and workflow management system stubs.

> "Self-describing workflows are partitions of a workflow that carry sufficient information to be managed by a local task execution agent, rather than a central workflow management system. A workflow management system stub is a light-weight component that can be attached to a task execution agent, which is responsible for receiving the self-describing workflow, modifying it and re-sending it to the next task execution agent."

Workflow templates are pre-planned workflow processes that are general enough to encompass several similar work processes. They can be instantiated by adding contextual data, other templates, or workflow fragments. A fragment is equal to a workflow stub.

[2]Object Relation Model
[3]Structured Query Language

3.7 Workflow Modelling

We have found UML 2 Activity Diagrams to be suitable to modelling workflow processes. Activity diagrams have in UML 2 been greatly expanded to support workflow modelling and provide these benefits:

- Decomposition. An activity diagram activity can be decomposed into subsidiary activity diagrams. This should make it easy to create sub-flows in the workflow process.
- Partitioning. It is possible to divide an activity diagram into partitions that shows the performer of the activities. The partitions can be divided into a two-dimensional grid.
- Signals. An activity can listen to signals, as well as send signals. This can be leveraged to provide modeling of context dependency.
- Flow final. A flow final makes it possible to end a thread of execution without ending the entire workflow process.
- Standard. The UML is a standard language. It is therefore a large amount of modelling tools created for helping modellers.

We will not go into more detail since workflow modelling evaluation is not part of our work. Fowler [17] gives a more detailed overview of the activity diagrams constructs.

3.8 Summary

To provide work support for mobile workers, we need a system capable of handling a set of activities with conditions determining when the activities should be executed. The set of activities needed, will continuously change when the mobile worker is doing his work. Workflow technology with an extension for adaptive workflow is therefore important to our work.

We will in PocketFlow use a combination of all the ideas explored in the previous section. This will be further explained in Chapter 11 and Chapter 12.

Chapter 4

Context-Aware Computer Systems

Weiser [60] presents a vision of a world where computers are integrated seamlessly into our environment. These computers are aware of their surroundings and can adapt their behaviour accordingly. Such **ubiquitous computing** is not feasible today, but improvements in technology related to embedded devices and sensor technology give rise to a more pervasive environment, supporting the development of context-aware applications. Lyytinen [30] presents an overview of the dimensions of ubiquitous computing in Figure 4.1.

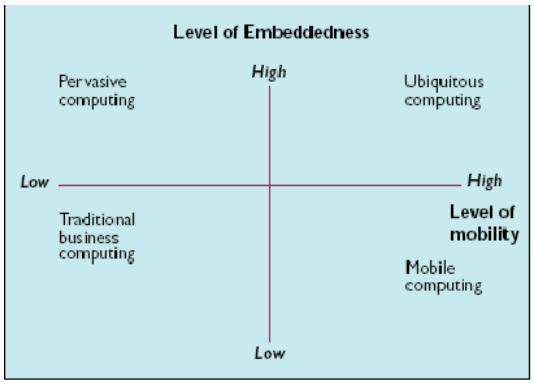

Figure 4.1: Dimensions of ubiquitous computing [30]

This figure presents the dimensions on making the computer invisible. Lyytinen proposes the main challenge in relation to ubiquitous computing to be the integration of large scale mobility with the pervasive computing functionality.

This chapter will refer to definitions of important terms related to context-awareness and presents some of the important challenges and benefits concerning the development of context-aware applications.

4.1 Context and Context-Classifications

A good understanding of context will help application designers utilise the possibilities of offering useful services and context-aware behaviour in applications. Relevant literature provides us with several definitions of context. Some authors provide examples to define context, while others use synonyms like the environment or situation of an entity. Dey and Abowd [14] believe these definitions are too specific, and propose the following definition of context:

> "Context is any information that can be used to characterise the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves."

The definition provided by Dey et al. is related to the development of the Context Toolkit [13], a framework consisting of toolkit components that can be combined to determine the contextual state of an application. Greenberg [20] argues that context is a dynamic construct, which means that in most cases it is difficult for designers of context-aware applications to list the set of contextual states that may exist, what information can accurately determine a contextual state, and what appropriate action should be taken from a particular state. He acknowledges the usefulness of Deys definitions and supporting framework, but emphasise how this is an elegant solution which only works to design context-aware applications for simple and highly routine contextual situations.

Our approach to context-aware computer systems is with the coordination of activities in adaptive, mobile work processes. For our purpose, we need to extend the definition of context provided by Dey and Abowd. Instead of focusing on the user and the interaction between a user and an application, our focus is on processes executing in an application environment. These processes involve activities, possibly competing for resources. In this setting, activities in the context-aware application's environment are the context. This approach to context may be related to activity theory presented by Nardi [36]. Nardi states the following about context in an activity system:

> "Activity theory proposes a very specific notion of context: the activity itself is the context. What takes place in an activity system composed of object, actions, and operation, *is* the context. Context is constituted through the enactment of an activity involving people and artifacts."

Chapter 5 gives an introduction to activity theory and situated actions.

The sentient object model [15] is presented in Chapter 7.1. This model presents a method to provide coordination of actions through the environment based on clients sensing their physical environment, and reacting to the sensed information according to a set of rules. As part of this model, definitions of context and context-awareness are provided.

A definition of context for the purpose of the sentient object model is [15]:

> "Any information sensed from the environment which may be used to describe the situation of a sentient object. This includes information about the underlying infrastructure available to the sentient object."

This definition focuses on coordination of sentient objects, operating without the need of an infrastructure.

Dey [14] defines the primary context types for characterising the situation of an entity as: location, identity, activity, and time. These primary context types can be used to acquire other relevant (secondary) information about the entity.

4.2 Context-Aware Computing

Context-awareness has become synonymous with the terms adaptive, reactive, responsive, situated, and context-sensitive in relevant literature. Previous definitions of context-aware computing can be categorised as focusing on using context or adapting to context [14]. When systems are limited to using context, the relevant services provided are sensing, interpreting, and reacting to contextual changes in the environment. Adaptive context-aware systems dynamically change and adapt their behaviour based on the context of the user and application.

Dey and Abowd [14] propose the following definition of context-awareness:

> "A system is context-aware if it uses context to provide relevant information and/or services to the user, where relevancy depends on the users task."

This is a general definition of context-awareness. It includes both systems that are using context and systems that are adapting to context as context-aware.

In the sentient object model, context-aware is defined as [15]:

> "The use of context to provide information to a sentient object, which may be used in its interactions with other sentient objects, and/or the fulfilment of its goals."

This definition is focusing on achieving coordination through the environment rather than provide information and services to the user.

When it comes to categorising features for context-aware applications, Dey and Abowd [14] propose a categorisation that combines ideas from previous taxonomies. The result is the following three categories of features that context-aware applications may support:

- Presentation of information and services to a user.
- Automatic execution of services.
- Tagging of context to information for later retrieval.

When working with aspects of activity coordination and adaptive workflow systems, we need to emphasise how these application features can provide benefits not only to users, but also to other (workflow) services. We therefore tailor these features for our process-centered approach to context, and include other services as well as users as the target of these features.

4.3 Challenges in Context-Aware Computing

Context-aware computing often involves the use of mobile devices, which have limited resources. Compared to static devices, mobile devices are small and suffer from limited power due to its size and weight. Data is transmitted through open airspace, making communication vulnerable to security violations. Wireless LAN, Bluetooth, and infrared communication are popular and in widespread use. The wireless connectivity is highly variable in performance and reliability.

Satyanarayanan [45] discusses some of the challenges in the emerging field of pervasive computing. For a computing system to be pervasive, it also has to be context-aware, and Satyanarayanan list the following challenges in the field of context-aware computing [46]:

1. How is context represented internally?
2. How is this information combined with the system and application state?
3. Where is context stored?
4. Does it reside locally, in the network, or both?
5. How frequently does context information need to be consulted?
6. What is the overhead of taking context into account?
7. What techniques can one use to keep this overhead low?
8. What are the minimal services an environment has to provide to make context-awareness feasible?
9. Is historical context useful?
10. What are the relative merits of different location-sensing technologies?
11. Under what circumstances should one be used in preference to another?
12. Should location information be treated just like any other context information, or should it be handled differently?

For our work, challenges to be especially aware of in this list are how context is stored (3 and 4), how frequently it has to be consulted (5), and what the overhead for taking context into account (6) is. These challenges relate to aspects of communication. An increase in context sources and workflow clients within a system constitute a severe challenge with regards to keeping the generation of messages at an acceptable level.

Our approach to context-aware computing is process-centered, which give rise to challenges related to context-aware process support. Sørensen et. al. [53] discuss some of these challenges in the research area of pervasive computing and workflow/process support with smart work processes.

Context-aware process support calls for a specification of pre/post-conditions related to process goals, and Sørensen et al. emphasise the need to develop models which facilitates integration between process models and the context-rich environment. A part of this integration involves creating protocols and semantics for information exchange between the real-world environment and the process environment. If order to achieve adaptation, specification of environmental behaviour related to an activity is important. There also has to be a specification of a uniform representation of sensors and actuators.

Another major challenge for context aware computer systems is related to the planning, specification, and performance of activities based on the environmental context. This involves managing process changes to ensure a consistent state of the process, and managing the dynamics of ad-hoc activities and process changes.

4.4 Advantages of Context-Aware Computing

Context-aware computing is about making our lives easier. There is a great potential for advantages by using context-aware features in real-life applications. For this to happen, research in this field must be able to make systems automate the collection of contextual information, and use this data to take the appropriate actions in the actual situations.

Smart work processes

Sørensen et. al. [53] introduce the notion of smart work processes. One of the main advantages of context-aware computer systems is the ability to provide work support for workers performing mobile work. Such smart work processes are sentient and adapt to relevant context by sensing the environment, perform context-based reasoning to reach process goals, and perform actuations which may change the workflow.

Potential for improved usability

When applications have available information about a user's context, it is possible to make several improvements in usability. The identity of a user can be detected automatically when needed, and available nearby resources can be presented to the user according to user-defined policies.

Automation of services

Context-aware computing can automate the execution of services for users and simplify everyday activities.

Increased safety

There are several industry scenarios that can benefit from increased safety by using applications that incorporate contextual information. E.g. by monitoring pressure, temperature, and other contextual parameters, applications can take actions to prevent accidents from happening.

Increased efficiency

Context-aware applications have a potential for increasing efficiency when implemented as workflow systems.

4.5 Summary

A good understanding of context, context-awareness, and the challenges related to these terms are important if we want to build systems with support for adaptive workflow. The use of implicitly sensed information allows the application's behaviour to be customised to the user's current situation. Definitions of context and context-awareness by Dey and Abowd help us understand what types of context, and what kind of context-aware features to incorporate in applications.

Although definitions provided by Dey and Abowd are helpful, they are not satisfactory with regards to the requirements we state for adaptive workflow systems in Chapter 9. We focus on processes and activities as important contextual information for use in workflow enactment, rather than viewing the user and the interaction between users and applications as the main focus. We therefore emphasise on how application features should be available to other (workflow) services as well as to the users of the application.

Chapter 5

Activity Theory and Situated Actions

This chapter investigates how aspects of activity theory and situated actions can be used to achieve situated planning within workflow systems and help us create a more dynamic workflow.

5.1 Situated Actions

Research in the area of cognitive science and artificial intelligence has created a large number of models for human behaviour. These models heavily rely on planning, and how humans use conscious plans to achieve the objective of an activity. Situated action models [37] are developed in contrast to such planning models of human actions. While models in cognitive science represent activities that are carefully planned, situated action models depend on the surroundings, and recognise the environment as an important shaper of activities.

Situated action models emphasises the way activities grow out of the particularities of a given situation, and therefore recognise that actions are always situated into a context, and impossible to understand without that context. With situated actions, the unit of analysis is the relation between the individual and the environment.

5.2 Activity Theory

Activity theory is a formal theory of human work activities and a philosophical framework for studying human work practises.

5.2.1 The Basic Principles of Activity Theory

The basic concept and unit of analysis in activity theory is the human activity, which has three main characteristics [5].

Activities are directed towards a material or an ideal object An activity is directed towards an object that motivates the activity. This object distinguishes one activity from another.

Activities are mediated by different artifacts Activity theory emphasise that human activities are mediated by tools as well as other artifacts.

Activities are social within a culture Human activities depend on the culture they are a part of.

Activities are composed of subject, object, actions, and operations [35]. The structure of activities can be represented as a hierarchy with three levels; activities, actions, and operations. Figure 5.1 shows the hierarchical levels of an activity [35].

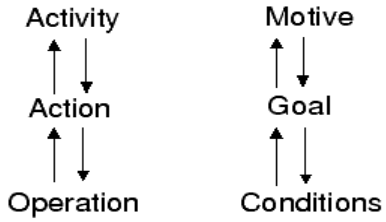

Figure 5.1: Hierarchical levels of an activity [35]

Activities are realised through goal-directed actions. Actions that are participating in activities have an immediate defined goal, and cannot be understood without reference to the corresponding activity. One activity may be realised through different actions depending on the situation. Actions are implemented through automatic operations. Operations do not have goals, but are used to adjust actions to the current situation. They are dependent on the conditions of the corresponding actions using them.

In the study of context and activity theory, Nardi [35] explains how activities themselves become context as stated in Chapter 4.1.

> "Activity theory, then, proposes a very specific notion of context: the activity itself is the context. What takes place in an activity system composed of object, actions, and operation, *is* the context."

5.2.2 Principles Derived from Activity Theory

Adams et al. [2] have derived ten principles from activity theory that help the understanding of work practises. These principles represent the authors' interpretation of the central themes of activity theory.

Activities are hierarchical This principle states that activities consist of one or more actions, and actions consist of one or more operations.

Activities are communal Activities are almost always communal. This means that activities often involve several participants working towards a common objective.

Activities are contextual The context of an activity deeply affects the way an objective is achieved.

Activities are dynamic Activities evolve asynchronously, and historical analyses are often needed to understand the current context.

Mediation of activity An activity is mediated by tools, rules, and division of labour.

Actions are chosen contextually Actions and operations are created, maintained and made available to any activity. Activities are then able to make contextual choices from the available actions and operations.

Actions are understood contextually An action's goal may be different from the objective of the activity in which the action is a component. An action may have a successful execution in reference to the overall objective of the activity.

Plans guide work Plans act as guidance of work processes and are modified in accordance to contextual information.

Exceptions have value Deviations from a plan can give rise to a learning experience which can incorporate additional exceptions into future executions.

Granularity based on perspective A piece of work may be an activity or an action depending on the perspective of the viewer.

5.3 Situated Planning

Bardram [5] gives the following definition of a plan based on activity theory:

> "A cognitive or material artifact which supports the anticipatory reflection of future goals for actions, based on experience about recurrent structures in life."

Traditional workflow systems have difficulties supporting unexpected changes and deviations from the original process model. These difficulties can be understood within the *planning paradox*; work is by nature ad-hoc and situated, but most organisations need pre-defined plans in order to organise their work. Bardram [5] approaches this paradox by viewing plans in relation to situated actions and activity theory. Both activity theory and situated actions emphasises the connection between plans and the contextual condition for realising these plans in actual work.

An important distinction is the difference between a plan and the instantiation of a plan. Plans are resources, and independent of concrete, situated activities. An instantiation of a plan applies the plan to a concrete problem where situated actions are adjusted to their contextual state. Since an instantiation depends on its contextual conditions, some actions will mirror the plan while others will not. After the completion of an instantiated plan, a comparison of the plan and its actual instantiation may reveal deviations that can give rise to learning.

Bardram [5] describes how situated actions are not to be viewed as the opposite of planning. Plans are made out of situated actions and realised in-situ. Situated planning involves planning while executing and the ability to modify a plan based on occurring events. Situated planning can therefore be characterised as the building, altering, sharing, executing, and monitoring of plans within cooperative work activities [5].

5.4 Relevance for Adaptive Mobile Work Processes

The theories described in this chapter are mainly HCI[1] research, but still useful for our work on adaptive mobile work processes and context-aware workflow systems. Both situated action models and activity theory provide help on understanding how work processes and workflow can be made more adaptive and support situated process and activity planning.

The situated action research provides a way of thinking about work as something that depends on the surroundings in addition to pre-defined plans. This view of actions which is situated into a context is relevant for our work on increased adaptivity for work processes.

Activity theory provides a division of work by abstracting an activity into hierarchical levels as shown in Figure 5.1. This representation of work provides a decomposition of activities into manageable pieces which facilitates more flexible work processes and also a more dynamic workflow.

In general, an understanding of situated planning as a combination of pre-defined templates using situated actions is useful when developing context-aware workflow systems. By using situated planning as a way of thinking may provide great benefits in relation to supporting context-aware work. Sørensen et al. [53] introduce the notion of smart work processes, which automatically adapt themselves based on reasoning on the contextual state of the environment. They recognise how situated actions and planning are a natural part of a support environment for smart work processes. By using the contextual state of the environment, activities can be created in-situ, either by the workflow client, or by the environment itself with an intelligent artifacts approach [51]. The potential benefits of situated planning are often related to increased safety and efficiency in the work environment.

5.5 Summary

When working with adaptive mobile work processes, plans are important, but not sufficient. Plans need to be instantiated based on information of the current situation. Both situated actions and activity theory may address issues in situated planning. Bardram [5] recognises this, and his conceptualisation of planning *in-situ* makes it possible to address planning in a way that does not require a rigid match between process models and actual work.

Using activity theory is a more formal approach to planning than situated actions. The hierarchical levels of an activity may be a good way of representing work.

[1] Human Computer Interaction

Chapter 6

Sensors and Actuators

Sensors are increasingly becoming smaller, more powerful, and smarter. This makes some of Weiser's visions [60] possible, and we will start this chapter with an introduction of smart dust, before we present wireless sensor networks and the sensors themselves. Actuators and intelligent artifacts will also be covered.

6.1 Sensors Like Dust and Brilliant Rocks

Satyanarayanan introduces in [47] the idea of sensors like dust, or **smart dust**. The idea involves smart sensors that can be produced in such volumes that they essentially do not have an individual cost. These sensors contain the actual sensing hardware, wireless connectivity, a small memory, a small CPU, and a small battery. When the sensors are deployed in their target environment they self-configure and react to the environment. For instance, if sensors are put into concrete when it is mixed, they can send data of the moisture level and temperature in the concrete. When the battery is depleted, the sensors are discarded since they have served their purpose. This type of sensors can be said to be immersive since they are embedded into the environment.

There is also a need for non-immersive sensors, like security cameras with image recognition. These sensors will be more expensive and hence not as easily deployed in large numbers. Satyanarayanan calls this type of sensors **brilliant rocks** since they have greater processing power and self- and environmental awareness.

6.2 Wireless Sensor Networks

Wireless sensor networks are self-organising, self-regulating, and self-repairing networks of wireless sensors nodes [12]. They combine processing, sensing, and peer-to-peer communications into tiny embedded devices [22]. The basic mode of operation for wireless sensor networks is different from traditional computer networks, due to their tight integration with the environment [44]. They are often deployed in large numbers and have to operate unattended for long periods of time. This makes it necessary to sacrifice some higher level functionality to make the sensor nodes as small and energy efficient as possible. To achieve higher level functionality, a hierarchy of sensor nodes is needed. The simple nodes connect

to higher level nodes operating as gateways. These gateway nodes provide access to larger networks and can also access even higher level nodes in the hierarchy as shown in Figure 6.1 [22].

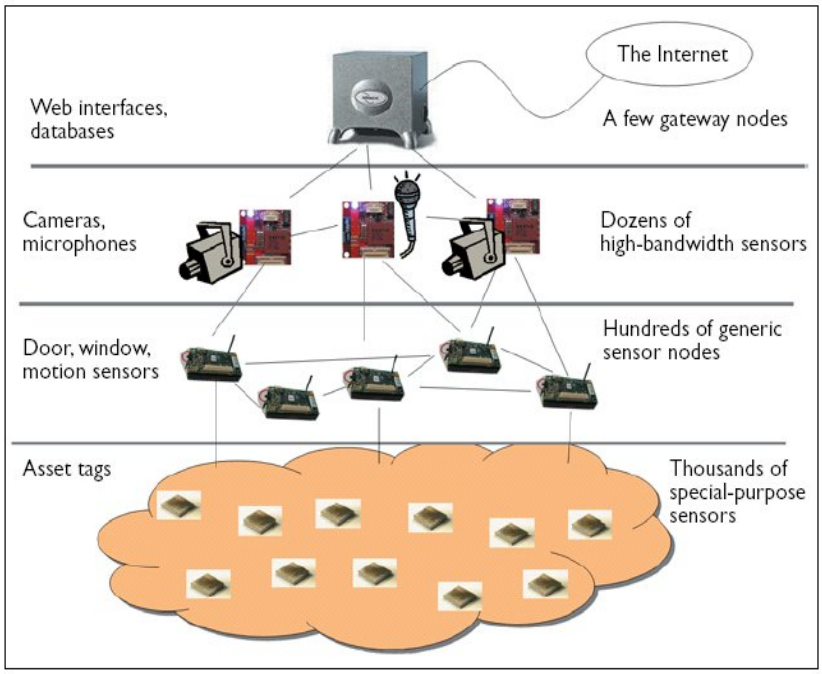

Figure 6.1: A sensor network divided into a hierarchy

In reference to Satyanarayanan, the lowest level of sensor nodes shown in Figure 6.1 is smart dust, while the two next levels can be considered to be brilliant rocks.

There are several challenges when deploying wireless sensor networks. Algorithms for routing in a hierarchical peer-to-peer network are very complex. Power must be used sparingly when the sensor node is running with a battery as the power source. The network must be extremely scalable and have a long-term performance that does not degrade over time. Additionally, sensor nodes may be exposed to harsh weather, theft, and malicious attacks. There exists several approaches to wireless sensor networks security [24], but since they are not important to our work we will not elaborate on them in this report.

6.3 SensorML

We will not use SensorML in our work since we do not focus on communication between the workflow client and sensors, but a short description is included since future work on PocketFlow could benefit from it. SensorML is a XML schema for defining the geometric, dynamic, and observational characteristics of a sensor and is developed by the Open Geospatial Consortium [59].

The purpose of SensorML is to:

1. Provide general sensor information in support of data discovery.

2. Support the processing and analysis of sensor measurements.
3. Support the geolocation of the measured data.
4. Provide performance characteristics (e.g. accuracy, threshold, etc.).
5. Archive fundamental properties and assumptions regarding sensors.

Although SensorML originally was intended for the Space Time Toolkit [58] it is general enough to be used in other implementations.

6.4 Actuators

An actuator is an object capable of performing some action. It may be a physical object like an engine, an application invocation, or something more abstract like a workflow activity. In the sentient object paradigm an actuator is defined as an entity that consumes software events, and reacts by attempting to change the state of the real world in some way via some hardware device. This may be the truth if one only considers physical devices to be actuators. In activity theory, an operation can be considered to be an actuator for an activity, in a workflow system an actuator may be an IT tool or manual work.

We will not discuss hardware since it is not relevant to our work.

6.5 Intelligent Artifacts

Strohbach et al. [51] presents the notion of intelligent artifacts. Figure 6.2 shows the components of an intelligent artifact.

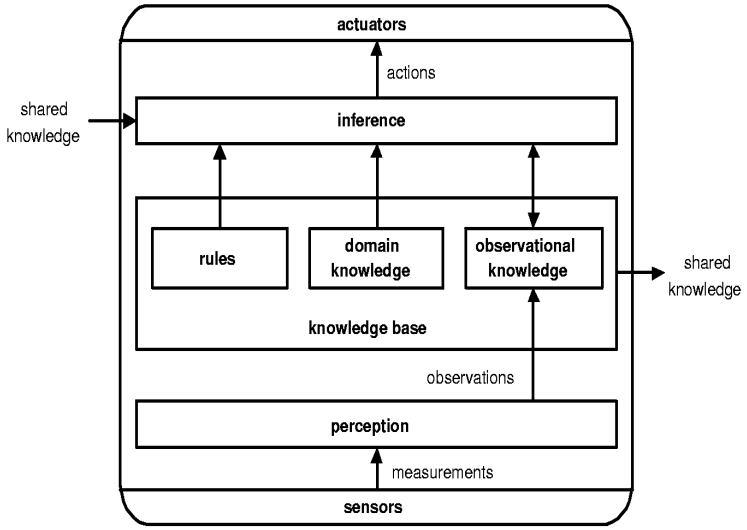

Figure 6.2: Components of an intelligent artifact

- **Sensors** This may be any of the sensor types discussed above.

- **Perception** Converts sensor data into observations meaningful to the application.
- **Knowledge base** Contains the domain knowledge of the artifact and dynamic knowledge about its situation in the world.
- **Inference** Processes the knowledge in the knowledge base and knowledge provided by other artifacts, and infer actions for the artifact to take.
- **Actuators** Actuate the inferred actions.

The knowledge base consists of facts and rules where facts are the foundation for decision-making and action-taking, and rules are used to infer further knowledge based on facts and other rule types. Strohbach et al. [51] categorise knowledge and rules in Table 6.1 and Table 6.2.

Domain knowledge	Domain knowledge built into the artifact, e.g. facts describing the physical nature of the artifact or general world knowledge.
Observational knowledge	Knowledge describing the situation of an artifact in the known world. It is based on facts that result from sensor-based observation.
Inferred knowledge	Knowledge inferred from previously established facts, which may be based on domain knowledge, observation, previous inference, and knowledge made available by cooperation artifacts.

Table 6.1: Knowledge stored in an intelligent artifact

Inference rules	Rules that describe inference of new facts from previously established facts.
Actuator rules	Rules that describe the facts that must be established in order to trigger an action.

Table 6.2: Rules of an intelligent artifact

An important aspect of intelligent artifacts is cooperation. Cooperation enables the artifacts to do cross-artifact reasoning and collaborative inference of knowledge. This is done by sharing knowledge in the artifact's knowledge base with other artifacts in what becomes a distributed knowledge base.

Since intelligent artifacts are autonomous entities with reasoning capabilities they can solve our problem of coordination between workflow participants. Integration of intelligent artifacts into PocketFlow is because of time constraints outside the scope of our work, but should be considered for future work.

6.6 Summary

Sensors which can be used in ubiquitous applications exist, but they are mostly used for research. Hill et al. [22] present the state of the technology today. We will in our work

emulate sensors and actuators, but the technologies described above will be very useful for creating realistic scenarios and test-beds for evaluating the proposed technology.

Chapter 7

Development Frameworks

This chapter presents frameworks which enable the development and testing of context-aware computer systems by providing a publish-subscribe middleware for event communication and an expert system for reasoning about context data.

7.1 CORTEX

CO-operating Real-time senTient objects: architecture and EXperimental evaluation (CORTEX) [57] is a research project and states its general objective as:

> "The CORTEX project will investigate appropriate architectures and paradigms for the construction of applications composed of collections of what may be called sentient objects - mobile intelligent software components that accept input from a variety of different sensors allowing them to sense the environment in which they operate before deciding how to react."

We will now briefly present the sentient object model and the CORTEX middleware to give an understanding of why the CORTEX project is relevant when developing context-aware computer systems.

7.1.1 The Sentient Object Model

Future pervasive environments may make it possible to develop applications consisting of mobile software components acting with a high degree of autonomy upon the environment via actuators based on input from sensors. The components have "intelligence" and co-operate via different network technologies. It is these components Fitzpatrick [15] terms **sentient objects**. Sentient objects interact using an event-based inter process communication. Figure 7.1 illustrates the sentient object model and its three main components; sensor capture and fusion, context representation, and inference engine.

The sentient object model provides the application developer with abstractions for sensors and actuators, and makes it unnecessary to do low level interaction with various hardware devices directly.

In the sentient object model, a *sensor* is defined to be [15]:

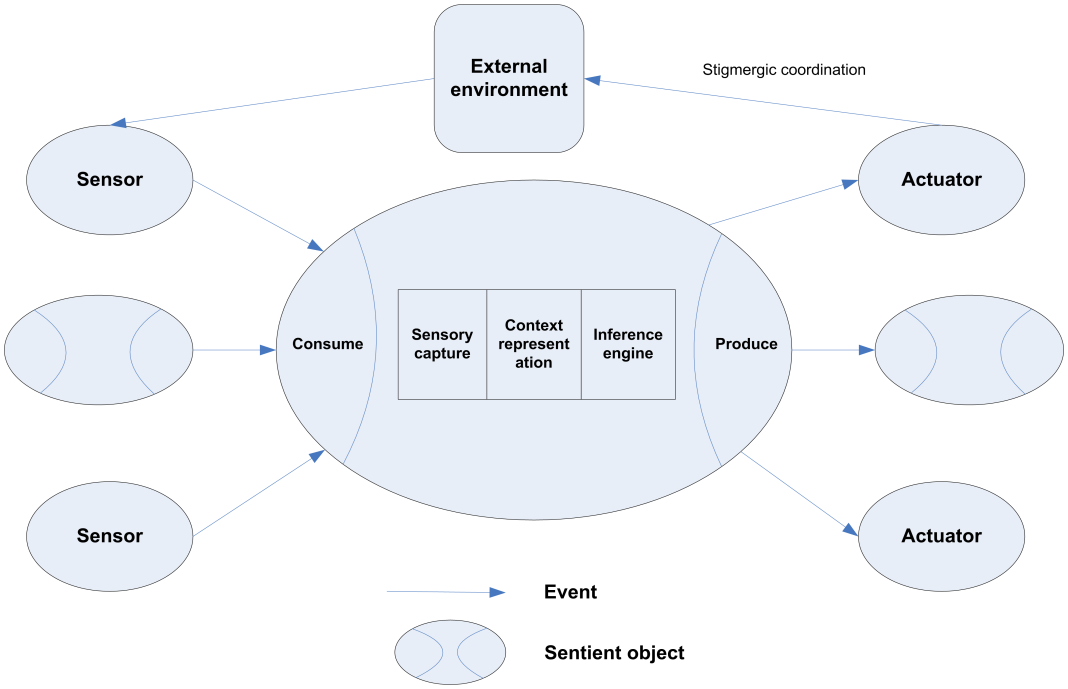

Figure 7.1: The sentient object model [63]

> "An entity that produce software events in reaction to a stimulus detected by some real-world hardware device."

An *actuator* in the sentient object model is defined as [15]:

> "An entity which consumes software events, and reacts by attempting to change the state of the real world in some way via some hardware device."

A *sentient object* is defined as [15]:

> "An entity that can both consume and produce software events, and lies in some control path between at least one sensor and one actuator."

Several characteristics of sentient objects are important when dealing with ubiquitous computing environments [7]:

Sentience Perceive the state of the environment.

Autonomy Operate independently in a decentralised manner.

Proactiveness Act in anticipation of future goals or problems.

Since the sentient object model relies on *event-based communication*, it is not dependent on centralised control.

The *sensory capture* component of the sentient object model receives input from sensors which needs to be integrated to determine the contextual state of the sentient object. Major issues to address in this area are data filtering and sensor fusion.

Context representation deals with the representation of contextual information in a way that is useful to the sentient object and can easily be exchanged amongst sentient objects. Raw sensor data often needs to be transformed before it may be considered useful information.

Sentient objects are context-aware in the sense that they interact with their environment with the help of sensors and actuators. An *inference engine* performs reasoning, and is the "intelligent" part of a sentient object. It is realised as an expert system with a knowledge base and a set of rules for deriving output.

7.1.2 CORTEX Middleware

Middleware can conceal programming complexity from application developers. CORTEX [63] has designed and implemented a flexible middleware for building dependable sentient computing applications. This middleware addresses several important research challenges related to context-awareness, use of communication models, QoS management, and routing in mobile ad-hoc networks.

To support a high level of configuration and reconfiguration, they have chosen to base their middleware on OpenCOM [10]. OpenCOM is a lightweight and efficient component object model based on COM.

Component Frameworks

A Component Framework (CF) is targeted at a specific domain, and has rules and interfaces that make sense in that domain. We briefly describe the publish-subscribe CF and the Context CF which provide a basis for building applications based on the sentient object model.

The **Publish-Subscribe CF** does not rely on any separate infrastructure. It supports a decentralised approach for discovering peers, for routing event notifications, and for event filtering. Events are represented as XML-based data, which provides interoperability and extendability of the event dialect. This is important since all subscribers should be able to interpret events without a priori knowledge.

Since the number of publishers and generated messages may be large, a subscription language is defined. The Filter Event Language (FEL) and its associated parser support subject filter and content filter. If a subscriber wants to receive all temperature related events, the FEL statement

```
//temperature/[]
```

is an example of a subject filter for which all temperature events are received. The statement

```
//temperature/%sensorName%=&room1&
```

is an example on a FEL statement for a content filter which only receives notifications on temperature events from the room1 sensor.

The **Context CF** is an implementation representing a sentient object and consists of two parts; sensor capture and fusion, and the inference engine. Sensor capture and fusion deals with sensor data and perform data fusion in order to manage uncertainty. The inference engine and its associated knowledge base is the "intelligence" of a sentient object. The inference engine used in the Context CF is based on CLIPS [1] [43]. CLIPS supports rule-based, object-oriented, and procedural programming paradigms. The Context CF uses the rule-based paradigm with rules that have an *if* and a *then* part.

A proof-of-concept demonstrator based on the notion of autonomous cooperating vehicles has been developed [63]. This demonstrator is using instances of the middleware CFs.

Since CORTEX middleware is based on the sentient object model, characteristics of applications which use this middleware are sentience, autonomy, and proactiveness [7], but also decentralisation and adaptivity [63]. These applications do not rely on any single central server, and have to cope with changing conditions during their lifetime. They will also have to interact with a physical environment while providing real-time services.

7.2 CLIPS

The C Language Integration Production System (CLIPS) [43] is a tool for building expert systems and is widely used both in industry and academia. It provides a complete environment for building rule- and/or object-based expert systems. CLIPS has been designed for integration with other languages such as C, C++, Java etc., but can also be run as a stand-alone tool.

Knowledge representation in CLIPS is provided through both heuristic and procedural paradigms. The heuristic paradigm represents knowledge as rules, which specifies actions to be performed for a given situation. For a rule to trigger, a set of conditions must be satisfied. These conditions are represented as facts. Facts and rules are used within a inference engine to derive the outcome when CLIPS is running.

A procedural paradigm is also supported in CLIPS for knowledge representation. New functions can be defined and called just like built-in functions of CLIPS. This gives the user a possibility to define new executable elements to CLIPS that perform useful effects.

7.3 Lua Language

Lua is a programming language created at the Computer Graphics Technology Group of the Pontifical Catholic University of Rio de Janeiro in Brazil. The language is created to provide scripting to extend conventional programming languages like C++. It is implemented in ANSI C and is considered to be very fast [40].

[1] C Language Integrated Production System

Lua can in our work be used to allow advanced queries, and to extend the functionality of the workflow enactment without having to modify the source code of the system.

Part III

Scenario

Chapter 8

Application Scenario - Aker Verdal

The main objective of this chapter is to present a relevant scenario for our work on adaptive mobile work processes based on a field study at the production unit of Aker Verdal [29]. Aker Verdal is a large construction site (800.000 square meters) and produces jackets and other marine concrete structures for offshore oil and gas production. Involved in the production are more than 700 workers and engineers. Chapter 8.1 gives an introduction to the work environment of Aker Verdal and covers how work is done today, while Chapter 8.2 covers our thoughts on possible future improvements to the work environment by creating a pervasive computing environment facilitating the implementation of adaptive, mobile work processes.

8.1 Current Work Environment

Figure 8.1 gives an overview of assembled steel jackets at the construction site of Aker Verdal. The weight of these steel jackets range from 1.000 to 20.000 tonnes and have heights ranging from 75 to 215 meters. Construction of these structures involves welding, cutting, and assembly of steel plates and tubes.

8.1.1 Work Processes

When modelling the production unit of Aker Verdal, our focus is on two work processes; a simplified steel plate workflow and a simplified structural tubular workflow.

8.1.2 Steel Plate Workflow

Figure 8.2 shows a simplified flowchart for the overall steel plate workflow. Steel plates are ordered from a manufacturer and processed at Aker Verdal for use in steel structures.

This work process consists of three main parts; pre-fabrication, pre-fabrication assembly, and final assembly.

Pre-fabrication is the first part of the steel plate workflow. Cutting information from a 3D-model is exported into cutting geometry which makes it possible to nest parts in a

Figure 8.1: Assembled steel jackets at Aker Verdal

nesting program, allocate steel plates and generate code for CNC-equipment[1]. Steel plates are then fetched from the warehouse and placed in a pipeline for automatic processing by a cutting robot. This cutting robot performs cutting according to data retrieved from the local area network. A human worker is supervising the operation. After cutting, the steel plate is transported to another pipeline for processing at a marker robot. The purpose of this marking is to point out where other steel components are to be placed on the plate. The marker robot operates automatically by using data retrieved from the Local Area Network (LAN) and is also supervised by a human worker. The marked steel plate is then transported for pre-fabrication assembly.

In pre-fabrication assembly, steel plates are again fetched from the warehouse and assembled on a flake[2]. Markings are added to the plates to ease the placement of parts from the pre-fabrication stage of the process. Relevant parts from pre-fabrication are then added which make the steel plate ready for the final assembly.

The last part of this workflow is the final assembly. The steel plates are transported for assembly at the main construction site and used in one of the steel structures produced at Aker Verdal.

There may be times in this workflow when parts are placed in the storage warehouse rather than being transported to the next processing entity right away.

8.1.3 Structural Tubular Workflow

Figure 8.3 shows a flowchart of the overall structural tubular processing workflow. Like steel plates, structural tubulars are also ordered from a manufacturer and processed at

[1] Computer Numerically Controlled - computer controlled machine tools for the purpose of manufacturing complex parts repeatedly

[2] A scaffold lowered over the side of a steel structure to support workers

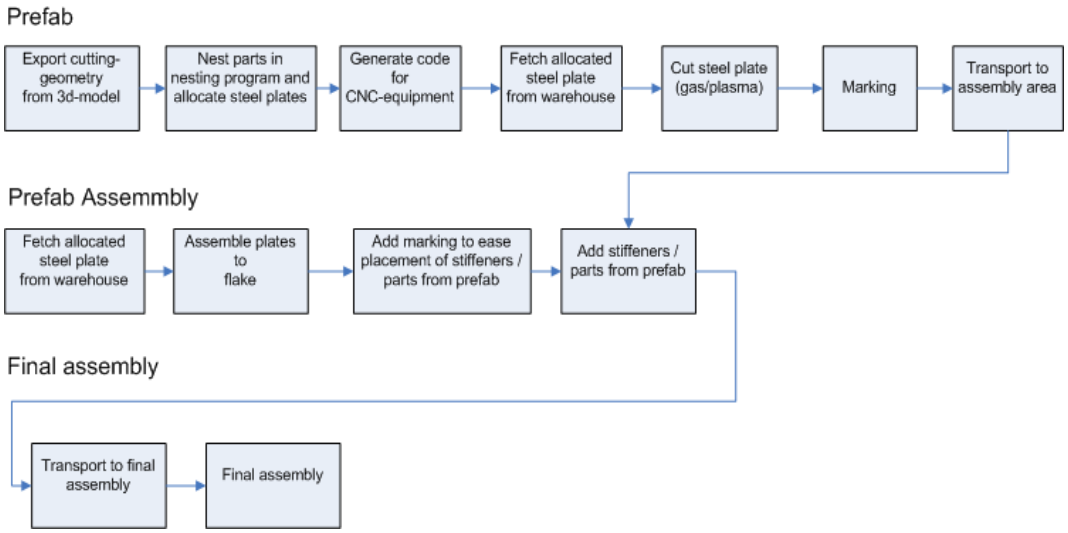

Figure 8.2: Simplified steel plate workflow

Aker Verdal for use in steel structures.

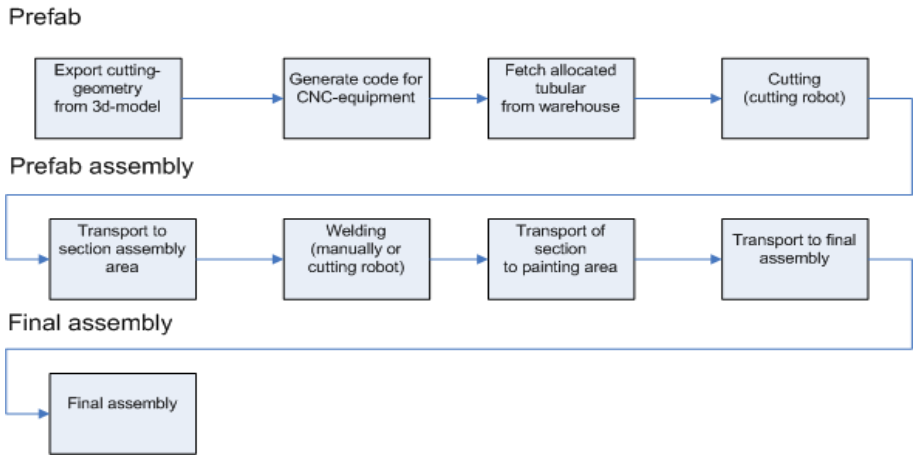

Figure 8.3: Simplified structural tubular workflow

Like the steel plate workflow, this work process also consists of three main parts; pre-fabrication, pre-fabrication assembly, and final assembly.

Pre-fabrication is the first part of the structural tubular workflow. Cutting information from a 3D-model is exported to cutting geometry and code is generated for CNC-equipment. Allocated tubular is fetched from a warehouse for processing by a cutting robot. The cutting robot performs the cutting and is supervised by a human worker.

In pre-fabrication assembly, the tubular is transported to the section assembly area. Welding is here performed by a welding robot. This welding is supervised by a human worker who also performs a visual inspection of the weld. The whole section is then transported

to the painting area for painting. The section needs to be painted to withstand corrosion and is painted by a painting robot. The painted section is then transported to the area for final assembly.

In the final assembly, the section is assembled into a large steel structure.

Between each of these processing steps there may be times when the tube is placed in storage rather then immediately being transported to the next processing entity.

8.2 Future Work Environment

Construction sites and shipyards such as Aker Verdal are hazardous work environments, which makes robots important as an addition to the existing workforce. This section proposes a future work environment for Aker Verdal by using mobile robots in a pervasive computing environment. These changes in the environment are not meant to be feasible in the near future, but are meant to act as a showcase for a future implementation of a system with support for mobile, adaptive work processes.

8.2.1 Mobile Robots

Robots can operate in hostile environments, performing work that is dangerous for human workers. The most common use of robots in construction of large steel structures is welding, but they may also be used in maintenance work, such as painting, cleaning, and inspections. Mobile robots can work on large structures by using tracks or as climbing/walking robots, sometimes alongside common workers. Figure 8.4 [3] shows a picture of a climbing and walking mobile robot used for welding and maintenance work.

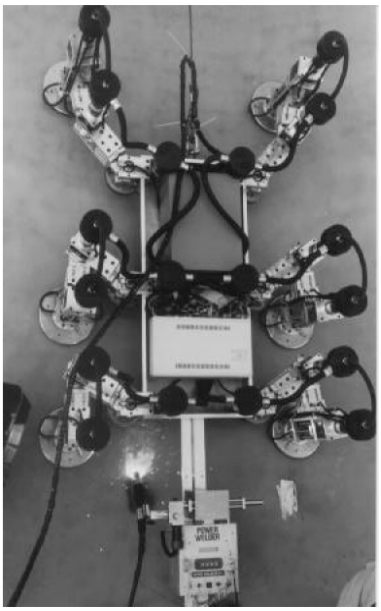

Figure 8.4: Mobile robot

There is a potential for increased efficiency in addition to the safety aspects. Altering complex robotic welding systems to handle single patterns is expensive. By using cooperating, mobile robots; reduced need for pre-planned activities leads to great savings in welding time [61].

To improve efficiency and safety, the worker is carrying a mobile terminal running a context-aware application which relies on tagging of relevant equipment in such a way that the context-aware application can identify its presence and use its information to automate and provide safe work processes. The mobile terminal is typically a robust PDA with WLAN support for identifying equipment.

Possible benefits of introducing mobile, cooperating robots in yards are [61]:

- Reduced risk of death or injury.
- Savings in welding times.
- Increased robotic welding accuracy.
- Reduced re-work rate.
- Reduced construction time.

These benefits relate to aspects of safety and efficiency.

8.2.2 Collaboration

Shipyard robots need to be mobile, and have some kind of coordination of their activities. Each mobile robot should be able to schedule work independently, and thus have some sort of "intelligence" supporting their execution of activities. Such mobile robots have a high degree of autonomy, and rely on advanced sensing technology for this purpose. One proposed solution is to use vision-guided robots to achieve automatic welding. This makes them able to do work without having a pre-programmed path, and thus saving time when setting up new complex welding jobs. The Wondemar [61] project suggests applications of such "intelligent" mobile robots to be: painting, welding, riveting, and in-situ cutting.

Intelligent Artifact Approach

These robots may use an intelligent artifact approach [51] to their collaboration. Intelligent, cooperative artifacts are able to asses their situation without the need of a separate infrastructure. The intelligent artifact approach relies on: embedded domain knowledge, perceptual intelligence, and rule-based inference. Strohbach [51] describes a set of components as an architecture of a cooperative artifact. These components will for this yard scenario be the following:

Sensing Each mobile robot needs to include sensor devices for observation of phenomena in the physical world, such as other robots or workers. This can e.g. be done by using robots with vision capabilities.

Perception This component associates sensor data with meaning in terms of the application. Location and sensor data observed by vision may provide meaningful terms used in the coordination of activities.

Knowledge base Contains the domain knowledge and dynamic knowledge about its situation. Domain knowledge for these mobile robots may be which robots exist in the environment and information about their welding. Dynamic knowledge may be observed robots and their location, as well as other information about cooperating artifacts.

Inference Knowledge and rules are used together to infer new knowledge about the world. Rules for mobile robots may for instance be about how to deal with situations where two robots compete for the same resource.

Actuators Actions that have been inferred are effected by actuators. Actuators for mobile robots may be different kinds of equipment for welding, painting, etc.

8.2.3 Impact on Human Work Processes

The future work environment described above have a strong impact on how human workers conduct their work. Work is to a greater extent automated via cooperation, and workers become more of a supervisor over equipment that ideally autonomically asses their own situation. A greater control over the situation on the workplace is achieved since the environment is feeding contextual information to workers and equipment. The mobile worker receives notifications of errors and warnings, and must make decisions in cases where the system is not capable of inferring a decision.

Increased control over the workplace includes increased efficiency and safety. Efficiency and safety are two of the main reasons for developing a context-aware work environment and should be the main impact on human work processes.

Part IV

Requirements

Chapter 9

Requirements for the PocketFlow Prototype

This chapter presents the functional and non-functional requirements for PocketFlow, and is partly based on the requirements presented by Nguyen and Nødtvedt [39] and in our depth project [21].

9.1 Functional Requirements

The functional requirements for an adaptive, ad-hoc workflow system are described in the next sections. These requirements are probably not complete since we only concentrate on the requirements specific for our approach.

9.1.1 Basic Workflow System

Here we present the requirements for a general workflow system. We only require a simplified workflow system for our work, without all the advanced features one would normally find in commercial workflow management systems. We will for instance not require external tools for modelling the workflow, monitoring it, etc.

Our workflow system must be able to enact one workflow process at a time. We will not allow multiple concurrent workflow processes on one client to prevent complexity. It is also not considered to be important to the applications built upon our architecture.

The workflow process will be loaded from an XPDL file (see Chapter 3.5.1). We will use a subset of the XPDL to reduce implementation time. The workflow process is divided into activities which may be split and joined by transitions to create a graph of execution (see Figure 3.1 for an example). Transitions may have a condition that is required to evaluate to true before the transition can be executed. When a transition condition does not evaluate to true, a transition exception must be sent to the correct exception handler. A workflow process also contains applications and/or participants that execute the activities.

The workflow system must also be able to communicate with workflow clients when activities are executed, and provide documentation when manual interaction is required. A workflow client can either be human or a system component.

The requirements for a simplified basic workflow system is summarised below.

F1 Adhere to the WfMC interface 1 meta-model. Support a subset of the XPDL process definition language.

F2 Interpret and enact a subset of the process definitions defined in the XPDL.

F3 Use activities completed by other workflow participants as precondition for own activities. In other words, waiting until another workflow participant has completed his activity before executing an activity.

F4 Evaluate conditions for transitions between activities by checking data fields in the workflow process.

F5 Perform enactment of a single workflow process or workflow process fragment. A single worker should not be executing several work processes concurrently.

F6 Provide workflow enactment feedback and information to the mobile worker. I.e. work lists and documentation.

F7 Support receiving feedback and information from the mobile worker.

F8 Communicate with other workflow clients in order to send and receive activities.

F9 Exception handling of transitions that do not have a satisfied condition.

F10 A workflow process must be able to execute external workflow applications.

9.1.2 Context-Aware Workflow System

Expanding the requirements for a general workflow system, this section adds support for context-awareness. A context-aware system needs to be able to find and connect to context sources, get information from context sources, and send information to context sources. The information must be filtered based on rules. This information is then used to provide conditions in activity transitions, as well as trigger the generation of new activities and transitions.

F11 Interpret and enact process definitions needed for context-aware workflows. Context information should be used in the evaluation of workflow transitions (precondition for activities). In other words, context information is used to provide data fields in the workflow process.

F12 Ad-hoc start of processes and activities based on context information.

F13 Filter sensor data, based on rules, to provide data the context client can use directly. This may include

- collection and aggregation of sensor data to provide an abstraction of the data. This makes it easier to use the data in the workflow process enactment.

- collection of sensor history for use in the workflow process enactment.
- documentation for the user. Such as procedures, blueprints, etc.

F14 Service for the discovery and look-up of context sources. Must be able to handle a large number of diverse context sources in a distributed and ad-hoc network.

F15 Support both polling and publish / subscribe mechanisms for context information retrieval from context sources, sensors, and actuators.

F16 Support the definition of context source look-up, polling, and conversion of the context information between the context representation and the workflow representation as XPDL elements. Rules and context source look-up information is stored in XPDL documents.

F17 Support the sending of actuation orders to actuators.

9.1.3 Adaptive Ad-Hoc Workflow System

By further expanding the requirements for a context-aware workflow system we add additional requirements for an adaptive ad-hoc workflow system. An adaptive ad-hoc workflow system must be able to use context information to generate new activities by instantiating workflow fragments. These fragments must be stored in a persistent storage and loaded on demand.

The workflow system must be able to receive activities and requests to remove activities from other participants of the distributed workflow process. For instance, security managers responsible for controlling the security aspects of the workflow may need to remove existing activities or add new activities to prevent dangerous situations from arising.

Furthermore, the workflow system should be able to handle unexpected situations and contextual states by performing situated planning with the help of exception handling. This may include creating new activities based on a set of rules, workflow fragments, context sources, and manual interaction. Manual interaction should enable the system to ”learn” new workflow process enactment paths.

The workflow system must also be able to coordinate its workflow with other workflow clients with conflicting interest to decide, based on predefined and ad-hoc rules, which workflow client gets priority. For instance, when two workflow clients require the same machine to be in two different states at the same time.

An adaptation can result in restructuring the workflow, either by inserting, deleting, modifying, redoing, or undoing activities.

F18 Perform distributed enactment of concurrent processes over several workflow clients that possibly compete for resources.

F19 Support template-based workflow enactment. This means instantiating general workflow processes by connecting to context-sources.

F20 Perform situated planning by supporting exception handling of undefined contextual states by creating new activities based on a set of rules, workflow fragments, context sources, and/or manual interaction.

F21 Support manual interaction from the mobile worker to create, modify, or remove activities.

F22 Provide persistent storage of rules, workflow fragments and templates, workflow history, and documentation for the user.

F23 Support central storage of workflow templates.

F24 Must be able to receive activities from other participants of the distributed workflow process.

F25 Must be able to coordinate the workflow process with other workflow clients with conflicting interest to decide, based on predefined and ad-hoc rules, which workflow client gets priority.

F26 Support re-validation of selected process paths, if the current path does not lead to the process goal. This means that if during the execution of a workflow process, it is discovered that the current context-based transitions do not lead to the process goal, the process may need to backtrack to a previous state and then re-validate the process path by checking context sources.

9.1.4 Mobility Requirements

A mobile worker requires a mobile device to be able to perform his work. This mobile device must be able to participate in heterogeneous networks while at the same time provide the mobile worker seamless communication. While this is not the focus of our work we include some general mobility requirements.

F27 Support for physical mobility and network mobility.

F28 Handling of unreliable communication. Unreliable communication should not halt the workflow process unless there are security issues involved.

F29 Support for disconnected operations and asynchronous communication. The workflow process must be autonomous when the mobile worker is outside the range of supported networks.

F30 Support for session mobility. The mobile worker should be able to seamlessly take his current workflow session from one network to another.

9.2 Non-Functional Requirements

9.2.1 Separation of Concerns

It is important to provide a clear separation of concerns between the modules of the architecture since PocketFlow will be used as a proof-of-concept application. This will

make it easier to provide loose coupling and promote reuse. Parnas [41] lists these benefits to modularisation and separation of concerns:

1. Managerial: Development time should be shortened because separate groups would work on each module with little need for communication.

2. Product flexibility: It should be possible to make drastic changes to one module without the need to change others.

3. Comprehensibility: It should be possible to study the system one module at a time. The whole system can therefore be better designed because it is better understood.

9.2.2 High Level of Responsiveness

The architecture must be able to respond quickly to changes in the environment. For instance when a dangerous situation arises, PocketFlow must be able to adapt quickly by for instance generating new activities. A high level of responsiveness is therefore important, and the architecture should support multithreaded services.

9.2.3 Connectivity

Our architecture also makes several assumptions with regards to the network connectivity.

Access to High-Quality Wireless Network

Our tests will be run on the standard 802.11b [54] WLAN technology.

Stable Connectivity

Since our tests will be run inside a building without much interference, the connectivity should be stable.

9.2.4 Component Harvesting

We make sure we do not "reinvent the wheel". There is no point in trying to create components from scratch when there are available components; either commercial, free, academic, or open source.

9.3 COTS Components

This section describes the COTS [1] components used in PocketFlow. Note that we do not limit COTS to only be commercial software, but also include open source, academic, and free software.

[1] Commercial Off The shelf Software

9.3.1 CORTEX Middleware

The CORTEX project presented in Chapter 7.1, proposes a middleware implementing the sentient object paradigm. The CORTEX middleware in PocketFlow provides us with a stable platform for ad-hoc communication between sensors, actuators, and workflow clients. It also enables us to create sensor and actuator services by using the event filter capabilities in the Publish-Subscribe Component Framework (CF).

The CORTEX Publish-Subscribe Middleware (PSM) can be deployed and registered on several PDAs. The middleware makes it easy to set up one PDA as a sensor, publishing events. Another PDA with PSM installed can then be set up to receive certain types of events by defining event channels and subscribe to them. PSM is, in other words, used to provide communication between workflow clients and between workflow clients and context sources.

The CORTEX middleware covers all connectivity requirements above.

9.3.2 CLIPS

CLIPS is an expert system which can easily be integrated with C/C++. A short introduction to CLIPS is given in Chapter 7.2. The CLIPS expert system fills our need for an expert system that can represent the contextual state of the environment with facts and provide reasoning over this representation based on rules.

CLIPS is initialised with a knowledge base stored in a file on the device. This knowledge base defines a production system consisting of rules and facts. With CLIPS integrated in our implementation, we can assert and retract facts and generate contextual-events based on rules that may trigger when the workflow client system executes.

The CLIPS component helps us implement requirements involving ad-hoc distributed workflow enactment and situated planning, as well requirements covering context as workflow process constructs.

9.3.3 Mobility and Context-Aware Workflow Prototypes

Nødtvedt and Nguyen have in their master thesis [39] developed several workflow prototypes in Java that incorporates contextual information caused by mobility. While we do not use their implementation directly we use the provided prototypes as inspiration for parts of our design.

9.4 Technical Constraints

We have identified several technical constraints for PocketFlow based on the requirements and COTS components. A technical constraint is a limitation in the architecture imposed by a component used in the design.

9.4.1 WfMC Reference Model

The WfMC reference model Interface 1 defines a standard for interconnection between workflow systems and is presented briefly in Chapter 3.5.1. Our workflow enactment

model must contain the elements in the XPDL standard to be able to use this reference model.

9.4.2 MS Embedded Visual C++

The CORTEX middleware is implemented as OpenCOM object created in embedded visual C++. Since we have decided to use the CORTEX middleware as a basis for our communication we are forced to implement PocketFlow in MS Embedded Visual C++ [33] since this is currently the only programming language for Pocket PC 2003 enabling access to the CORTEX middleware COM[2] objects.

9.4.3 PocketPC 2003SE Handheld Device

A Fujitsu Siemens handheld device is used for deployment of the implementation. This device supports wireless technologies such as Bluetooth / WLAN, and is shipped with the Microsoft Pocket PC 2003SE operating system. We must therefore make sure we only use functionality available on this platform.

9.5 Discussion

A system with full support for all the listed requirements is outside the scope of this thesis, and we therefore concentrate on the requirements for an adaptive context-aware workflow enactment service. Requirements covering cooperation are not considered to be the focus of our work. We do not provide much discussion on requirements covering context-source discovery and abstraction since this is already covered by the CORTEX middleware.

[2]Component Object Model

Chapter 10

Use Cases

This chapter describes the use cases for the PocketFlow arhitecture. These use cases are high-level descriptions of how PocketFlow performs workflow enactment and interacts with other actors.

10.1 PocketFlow Use Case Overview

Figure 10.1 shows an overview of a mobile worker, using the the PocketFlow system to enact a workflow. The workflow enactment involves communication with sensors, actuators, and the Publish-Subscribe Middleware (PSM) to send and receive events. It also involves communication with other workflow clients to send and receive activities. The use cases presented in this chapter is meant to provide a high level understanding of the PocketFlow system. Parts of the PocketFlow system is therefore considered to be actors in the use cases. The *Do mobile work* use case is very general in nature since our focus is on adaptive ad-hoc workflow systems and not standard workflow systems.

10.2 Use Case: Enact Workflow

This use case presents the overall enactment of a workflow process in PocketFlow along with possible extensions to the basic path.

Actors: Mobile worker, sensor, actuator, PSM (Publish-Subscribe CF middleware), and remote workflow client.

Trigger: Mobile worker downloads a work order to his PDA.

Main success scenario:

1. The use case starts when the mobile worker downloads a work order to his PDA.
2. PocketFlow loads the workflow process template.
3. Use *Sensors and Actuators Setup* (Chapter 10.2.1).
4. PocketFlow starts the workflow enactment.

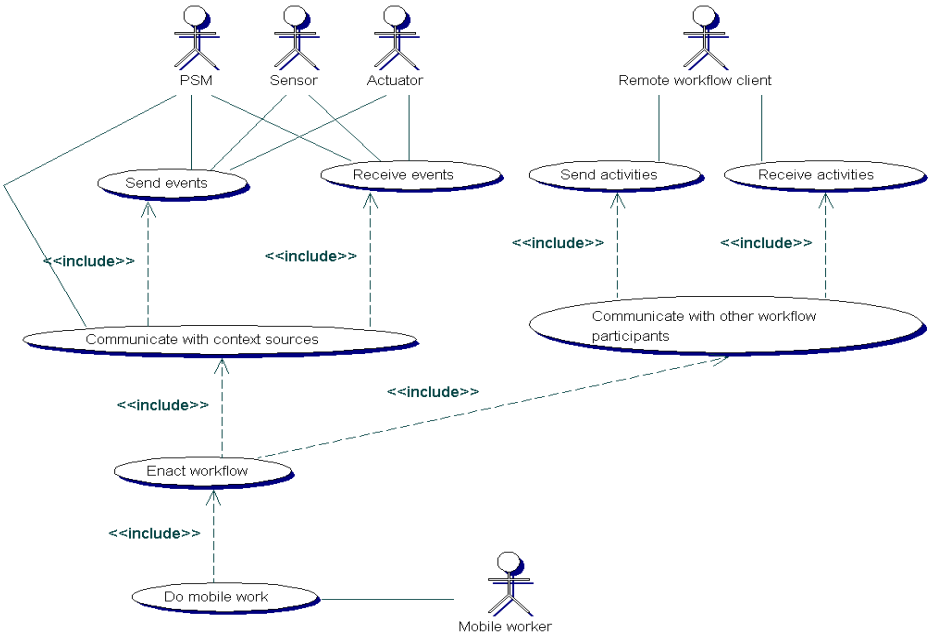

Figure 10.1: PocketFlow use case overview

5. PocketFlow runs the workflow enactment.
6. PocketFlow terminates the workflow enactment when the workflow process goal has been reached.

Extensions:

5 a) PocketFlow receives a sensor update from a sensor.

1. PocketFlow updates the knowledge base.
2. PocketFlow updates corresponding data field in the workflow process.
3. Use *Generate New Activities* (Chapter 10.2.2).
4. Use *Remove activities* 10.2.3.

5 b) PocketFlow derives the need to send an actuation order.

1. The PSM sends the actuation order to the correct actuator.

5 c) PocketFlow encounters an enactment exception.

1. PocketFlow determines the correct exception handler for the exception.
2. The exception handler inserts the exception in the knowledge base.
3. Use *Generate New Activities* (Chapter 10.2.2).
4. Use *Remove activities* 10.2.3.

5 d) PocketFlow receives an activity from a workflow participant.

1. PocketFlow updates the workflow process.

5 e) PocketFlow needs to provide feedback to the mobile worker.

1. PocketFlow display information on the PDA screen to the mobile worker.

5 f) Mobile worker has completed an activity.

1. PocketFlow marks the activity as completed and continues workflow enactment.

5 g) PocketFlow needs to send an activity to a workflow participant.

1. If the workflow participant is the mobile worker then
   (a) PocketFlow sends the activity to the mobile worker via some user interface.
   end if
2. If the workflow participant is a remote workflow client then
   (a) PSM sends the activity to the remote workflow client.
   end if

5 h) PocketFlow stops the workflow enactment. If the state required by the stopping event is obtained then

1. Continue step 5.

end if

10.2.1 Use Case: Sensor and Actuator Setup

Sensor and actuator set-up for PocketFlow involve using the PSM to setup event channels and subscribe to specified sensors.

Actors: PocketFlow, sensor, and actuator.

Trigger: PocketFlow needs to instantiate a workflow process template.

Main success scenario:

1. The use case starts when a workflow process template needs to be instantiated.
2. The PSM part of PocetkFlow looks up sensors and actuators in range.
3. The PSM part of PocetkFlow subscribes to the specified sensors and actuators.

Extensions:

2 a) PSM can not look up specified sensors or actuators because they are not in range.

1. PocketFlow creates an enactment exception.

10.2.2 Use Case: Generate New Activities

The workflow may be changed during enactment. New activities are generated when needed, and integrated with the workflow.

Actors: PocketFlow and remote workflow client.

Trigger: PocketFlow knowledge base is updated.

Main success scenario:

1. The use case starts when the expert system is run after a knowledge base update.
2. If the expert system infer a need to generate new activities based on the knowledge base then
   (a) PocketFlow generates new activities.
   (b) PocketFlow integrates the activities with the workflow process.
   (c) PocketFlow notifies listening workflow clients of the new activities.
   (d) Remote workflow clients receive the notification and act as required.

   end if

10.2.3 Use Case: Remove Activities

The workflow may be changed during enactment. Activities are removed when needed, and changes are integrated with the workflow.

Actors: PocketFlow and remote workflow client.

Trigger: PocketFlow knowledge base is updated.

Main success scenario:

1. The use case starts when the expert system is run after a knowledge base update.
2. If the expert system infer a need to remove activities based on the knowledge base then
   (a) PocketFlow removes activities.
   (b) PocketFlow integrates the changes with the workflow process.
   (c) PocketFlow notifies listening workflow clients of the removed activities.
   (d) Remote workflow clients receive the notification and act as required.

   end if

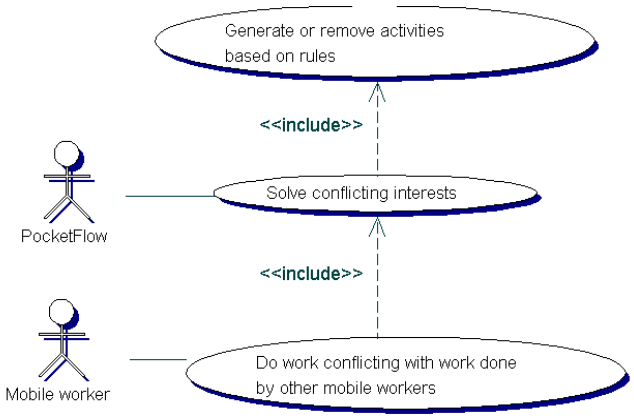

Figure 10.2: Cooperation use case

10.3 Use Case: Cooperating Workflow

Cooperating workflow clients are described in Figure 10.2.

Actors: Mobile workers and PocketFlow.

Trigger: Two or more mobile workers are located in the same area with conflicting process goals.

Main success scenario:

1. The use case starts when two mobile workers enter the same location with conflicting process goals.
2. Use *Generate new activities* 10.2.2.
3. Use *Remove activities* 10.2.3.

Extensions:

1 a) PocketFlow derives a need for coordination because two or more mobile workers compete for resources.

1 b) PocketFlow derives a need for coordination because a dangerous situation is detected (safety).

1 c) PocketFlow derives a need for coordination because two or more mobile workers starts doing the same work.

Part V

Architecture and Design

Chapter 11

The PocketFlow Architecture

11.1 Introduction

This architecture is a continuation of the work started in [21], and is meant to provide a common understanding of the problems surrounding adaptive and mobile work processes. The architecture is intended to provide a basis for a complete system, and as such the design in Chapter 12 is simplified. We will clearly state when there has been a simplification.

11.2 Architectural Drivers

We use architectural drivers to define what we want to achieve with our architecture. The following architectural drivers are chosen to be the main focus of our architecture.

11.2.1 AD1 - Decentralised Workflow Management

Workflow clients provide local workflow enactment based on contextual information from the environment. This includes support for workflow templates and fragments (see Chapter 3.6.3). Coordination of work can also be performed decentralised, and should optimally include all actors capable of reporting their progress.

11.2.2 AD2 - Autonomous Workflow Clients

All workflow clients can consume and react to events from the environment without relying on centralised control.

11.2.3 AD3 - Event Based Asynchronous Communication

The CORTEX Publish-Subscribe middleware supports event based asynchronous communication between workflow clients and the environment. In other words, it does not rely on the presence of any separate infrastructure, and can be used in an ad-hoc mobile network.

11.3 Stakeholders

Since this is an academic research project, the number of stakeholders is rather limited.

- **End user** While we have created an architecture to support the scenario described in Chapter 8, we do not have an actual end user since this is a proof-of-concept architecture.
- **Our self** We are implementing the architecture in our master thesis. (Developers and Acquirers)
- **MOWAHS** May acquire the architecture and improve on it in the future. It is also conceivable that MOWAHS may use it as a proof-of-concept, and as a test-bed for new research. (Users and Maintainers)

11.4 High-Level Architecture Overview

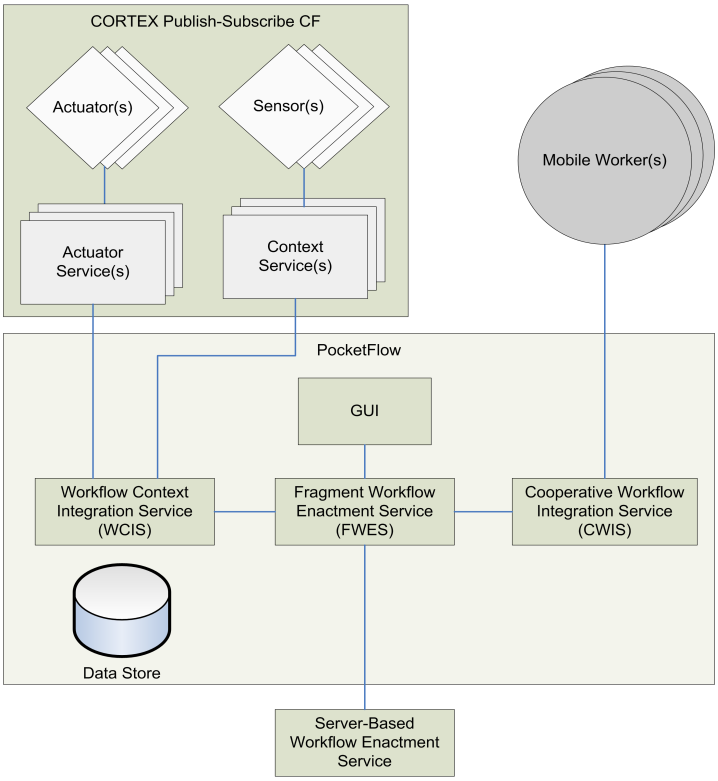

Figure 11.1: Architecture overview

An overview of our architecture is presented in Figure 11.1. A high-level description of each of the components in this figure is presented below.

11.4.1 Mobile Worker

A mobile worker is an entity that uses a workflow client to cooperate with other mobile workers. The entity may be a person or a machine, but for our purpose we assume that it is a human worker.

11.4.2 Workflow Client

The workflow client is the computer system running the workflow enactment service and the services it depends on. For a human worker this computer system would typically be a PDA while a machine could use a wider range of computer systems.

11.4.3 GUI

The graphical user interface gives information to, and receives commands from, a human user. The GUI must be adapted to a small screen size. A more advanced implementation of this architecture could potentially integrate other methods of communication between the user and the workflow client, but that is outside the scope of our work.

The GUI component covers the requirement for feedback to user on workflow enactment status.

11.4.4 Fragment Workflow Enactment Service (FWES)

The fragment workflow enactment service is an extension of a standard workflow enactment service that can execute workflow process fragments in addition to standard workflow processes.

The FWES consists of two parts; A workflow enactment service and a XPDL parser. The workflow enactment service allows heavy modification of the workflow process at run-time. The XPDL parser is able to parse incomplete XPDL documents (workflow fragments) and insert them into the workflow enactment service. See Appendix E for a XPDL template example.

An implementation of this component should cover the requirements for a basic workflow system (Chapter 9.1.1), except the requirement for feedback to the user on workflow enactment which is covered by the GUI component and the requirements for communication with other workflow clients which is covered by the WCIS component.

11.4.5 Workflow Context Integration Service (WCIS)

This service integrates the FWES with the context-services and actuator-services. It is responsible for look-up of, and subscription to, sensors and actuators, receiving context events from sensors and actuators, sending actuation request to actuators, handling transition exceptions in the FWES, and creating new activities based on rules.

An implementation of the WCIS component should cover some of the requirements we have specified for a context-aware workflow system (Chapter 9.1.2). Requirements for being able to interpret and enact process definitions needed for context-aware workflows and ad-hoc start of processes/activities based on contextual information are covered with support from the FWES component.

11.4.6 Data Storage

The data store provides persistent storage of workflow templates and fragments, pre- and post-condition rules, history, and enactment information.

This component should cover requirement for persistent storage of rules, workflow fragments, templates, workflow history, and user documentation for easy retrieval.

11.4.7 Cooperative Workflow Integration Service (CWIS)

The cooperative workflow integration service is responsible for the management and coordination of activities in a multi-actor environment. Coordination is based on policies, which enables the workflow enactment service to send messages about coordination needs, competitive resources etc. to all involved parties. Relevant coordination can for the workflow enactment service be to send the state of the currently executing activity, and then possibly prepare other users for new activities to be started within a time limit. Other coordination messages may be communication with other users, including the need for artifacts, services, actuators, or a preferred environmental state. The cooperative workflow integration service can also send contextual events to workflow clients based on coordination reasoning. These events may include information about the need for artifacts, services, actuators, or an environmental state with priorities concerning demands of different kinds. The coordination of the client workflow enactment service may derive new process fragments which can be distributed to other clients based on their needs.

In an implementation, this component should be integrated with the WCIS since they share functionality, but we have separated them for readability.

This component covers many of the requirements we have specified for an adaptive, ad-hoc workflow system (Chapter 9.1.3). Relevant requirements relate to distributed enactment of concurrent processes over several workflow clients, support for situated planning by having exception handling of undefined contextual states, the possibility of receiving activities from other participants of the distributed workflow process, and coordination of workflow clients with conflicting interests.

11.4.8 Context Service

A context service manages sensors and provides contextual events to context clients according to their rules and preferences. When a context service receives an event from a sensor it will filter it based on the context client's rules and preferences, provide reasoning on the sensor data, and convert it into something the context client can use.

A context service can use one or more context services other than itself to discover and manage sensors by either subscribing to the context services as a context client, or by downloading the other context services' lists of context sources. Context services can be realised as software run on a user client, or as physical devices in the work environment.

This component should cover requirements for the discovery and look-up of context sources and the support for polling and publish-subscribe mechanisms for context information retrieval.

11.4.9 Sensor

A sensor is an electronic sensor, capable of sending sensor data digitally. It has a simple mechanism for marshalling SensorML data, in addition to support for broadcasting data periodically, or receiving poll requests from a context source. Sensors may vary from very simple, to smart and autonomous entities.

This component should cover requirement related to support for polling and publish-subscribe retrieval of context information.

11.4.10 Actuator Service

An actuator service is responsible for look-up of actuators, validation of actuation orders, and initiating actuation orders in an environment. For instance, by starting/stopping an engine, executing IT tools, sending messages, etc.

Like the Context Service, this component should cover requirements for the discovery and look-up of context sources and support polling and publish-subscribe mechanisms for context information retrieval. The requirement for sending actuation orders should also be covered by this component.

11.4.11 Actuator

An actuator is responsible for executing actuation commands received from an actuator service. Actuators may be physical devices, applications, or other workflow enactment services.

Like the sensor, this component should cover requirements related to support for polling and publish-subscribe retrieval of context information.

11.4.12 Server-Based Workflow Enactment Service

A central workflow enactment service is responsible for managing the complete, high-level workflow process for all the involved participants. This involves sending workflow processes to users based on plans. These workflow processes are templates which are instantiated by the workflow client by using context information from the environment.

This component should cover the requirement for central storage of workflow templates.

11.5 The Sentient Object Model and the Intelligent Artifact Paradigm

The PocketFlow architecture uses ideas and concepts from both the sentient object model and the intelligent artifact paradigm.

11.5.1 Sentient Object Model

The WCIS component of the PocketFlow architecture described in Figure 11.1 shares several similarities with the sentient object model described in Chapter 7.1.1. Both the

PocketFlow architecture and the sentient object model receive events and can possibly trigger responses as actuations to the environment. The state of the environment is represented in a knowledge base on the workflow client.

11.5.2 Intelligent Artifact Paradigm

The architecture described in Figure 11.1 also has several similarities with the intelligent artifact approach described in Chapter 6.2. The intelligent artifact architecture has a knowledge base consisting of rules, domain knowledge, and observational knowledge. For the PocketFlow architecture, the WCIS and CWIS components creates the domain knowledge during the set-up of a new workflow process (context-sources etc.), while context updates are a kind of observational knowledge which is added to the knowledge base during execution of the workflow. The intelligent artifact approach uses a distributed knowledge base to enable cross-artifact reasoning. This feature is not supported in the PocketFlow architecture, but the CWIS will to some extent be able to coordinate different workflow clients based on predefined and ad-hoc rules.

11.6 Limitation of Scope

The next chapter presents a more detailed design of these architectural components. Our focus is on the FWES and the WCIS components, and while the CWIS component is important for a fully working prototype, it will not be examined in detail because of time constraints.

Our ambition for development of this architecture is to reach a satisfactory implementation of the FWES and the WCIS components which is usable for testing with workflow processes and contextual information. A full scale implementation of this architecture requires work far beyond the scope of our master thesis.

Chapter 12

Design

In this chapter we present a more detailed description of the components in the architecture. All diagrams are UML diagrams. We will only describe the most important classes and packages in the design. For further documentation, see Appendix I.

The packages in the workflow client are ordered as follows:

```
FWES
    application
    EnactmentRepresentation
    exception
    scripting
WCIS
    application
utils
```

12.1 Design Patterns

We have in our design used some common design patterns. In this chapter we will give a brief description of each pattern.

12.1.1 Model View Controller Pattern

The model view controller pattern [52] is based on separating software into three distinct parts.

- Model: The model contains any business logic needed in the application.
- View: The view contains classes responsible for rendering the user's view.
- Controller: The controller classes works as a connection between the model and the view.

From this pattern we get a clear separation of concerns since the model does not know anything about the view. If the view or the model changes, only the controller needs to be changed.

When designing PocketFlow we used this pattern to provide context integration and for displaying the GUI. In both cases the workflow client component is considered to be the model.

12.1.2 Dependency Injection Pattern

To further ensure that we have loose coupling and clear separation of concerns, we base the design on the dependency injection pattern [16]. The dependency injection pattern is based on the idea that there is a separate object responsible for assembling the dependent objects in the application as shown in Figure 12.1. By using interfaces, these objects do not know of the implementation of other objects.

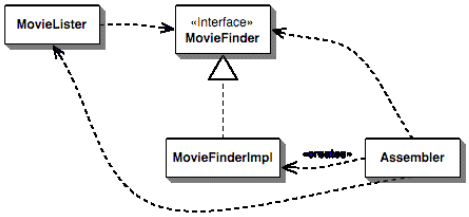

Figure 12.1: The dependency injection pattern [16]

All parts of the workflow enactment service requiring a connection to other components are in our design defined by interfaces. The workflow enactment only uses these interfaces in the enactment. The application implementing the workflow client must then specify the implementation of those interfaces by injecting the classes into the workflow enactment service.

12.1.3 Observer Pattern

The purpose of the observer pattern [18] is to define a one-to-many dependency between objects so that when one object changes it's state, all it's dependents are notified and updated automatically. We use this pattern in the workflow process executor in Chapter 12.2.7. The workflow process executor launches a thread that observes the workflow enactment with activities and transitions working as subjects to the observer thread.

12.1.4 Singleton Pattern

The purpose of the singleton pattern [18] is to ensure that there exists one and only one instance of a class in the application process. This makes it both easy to get an instance of the class, and to prevent duplication of the instance.

12.2 FWES Package

This is an implementation of the FWES component described in the architecture. Figure 12.2 shows a diagram of the most important classes in the fragment workflow enactment service and the relationships between them.

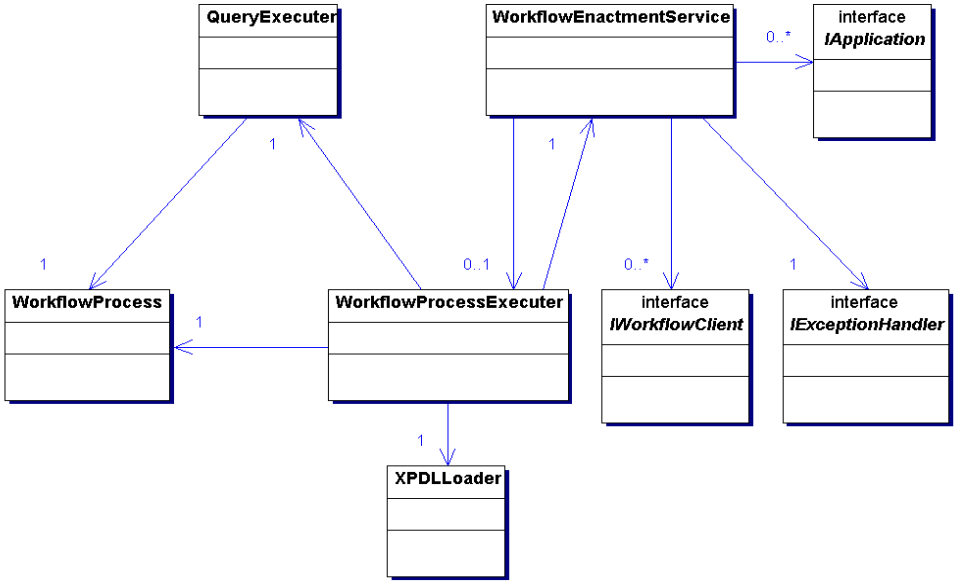

Figure 12.2: FWES class diagram

12.2.1 Note on Thread of Execution

We will use the term **thread of execution** to describe a single linear path of activities and transitions in the graph of execution as described in Chapter 3.2.

12.2.2 IApplication Interface

This is an interface all workflow applications must implement. A workflow application must be registered with the workflow enactment service before the workflow process is started if it is to be used.

The interface forms the basis for calling external applications from the workflow. We will therefore use workflow applications to implement context-awareness in the workflow enactment as shown in the *Sensor and Actuator Setup* use case in Chapter 10.2.1. This is an implementation of requirement F10, and is used as a means for communication from the FWES component to the WCIS component in the implementation of requirements F14, F15, F16, F17, and F19.

12.2.3 IWorkflowClient Interface

This is an interface all workflow clients must implement. A workflow client must be registered with the workflow enactment service before the workflow process is started if it is to be used. Workflow clients can be listeners, performers, or both. If the workflow client is of the listener type, it will receive a notification when an activity is starting and when an activity has completed. A performer workflow client will receive an activity for it to perform. This activity must then be completed before the thread of execution can continue. Both listener and performer workflow clients are notified of workflow execution status change.

The *IWorkflowClient* interface is a direct implementation of requirements F6, F8, and as a communication means from the FWES component to the WCIS component in requirement F25.

12.2.4 IExceptionHandler Interface

The exception handler must implement the *IExceptionHandler* interface. An exception handler is responsible for interpreting exceptions in the workflow enactment. There can exist one and only one exception handler in the workflow enactment service. The current implementation will only send a transition exception to the exception handler.

This interface implements requirement F9 and partially requirement F20.

12.2.5 QueryExecuter Class

The query executor is responsible for evaluating transition conditions. Conditions are specified as simple Lua script expression [40] on the form:

```
getDataFieldValue( "<data field name>" ) <equality operator> <value>
```

where <data field name> is any data field in the current workflow process, <equality operator> is any of "==", "~=", "<", ">", "<=", or ">=", and <value> is the desired value of the data field.

This class helps the WorkflowProcessExecuter in implementing requirement F4.

12.2.6 WorkflowEnactmentService Class

The workflow enactment service contains zero or one executing workflow process, and works as an interface between workflow clients and applications, and the workflow process executor. It supports starting, pausing, resuming, and stopping the workflow enactment.

When the workflow enactment service receives

- an execution request to an application from the workflow process executor, it looks up in the application registry and sends the execution request to the correct application.
- a status update, activity starting, or activity ended notification from the workflow process executor, it will look up in the workflow client registry and send the notifications to all the workflow clients marked as workflow listeners.
- a register activity event from the workflow process executor it will look up the correct workflow client and register the activity with it.

The *WorkflowEnactmentService* class also function as the main accesspoint to the workflow enactment for workflow clients.

The workflow enactment service works as an interface from the WCIS and GUI components to the FWES component in requirements F7, F8, F12, F18, and F24 and in the *Enact workflow* use case in Chapter 10.2.

12.2.7 WorkflowProcessExecuter Class

The workflow process executor is responsible for the enactment of a single workflow process. It launches in a separate observer thread (see Chapter 12.1.3) so the workflow enactment service may continue to be responsive.

Note that the workflow process executor will not enact all process constructs defined in the XPDL. We only support activities with implementation and route type of activities. Block activities and subflows are not supported. We further restrict the split and join transition restriction to only be of the "AND" type. We do not support the procedure type implementation of the tool type. See [55] for more information about these types.

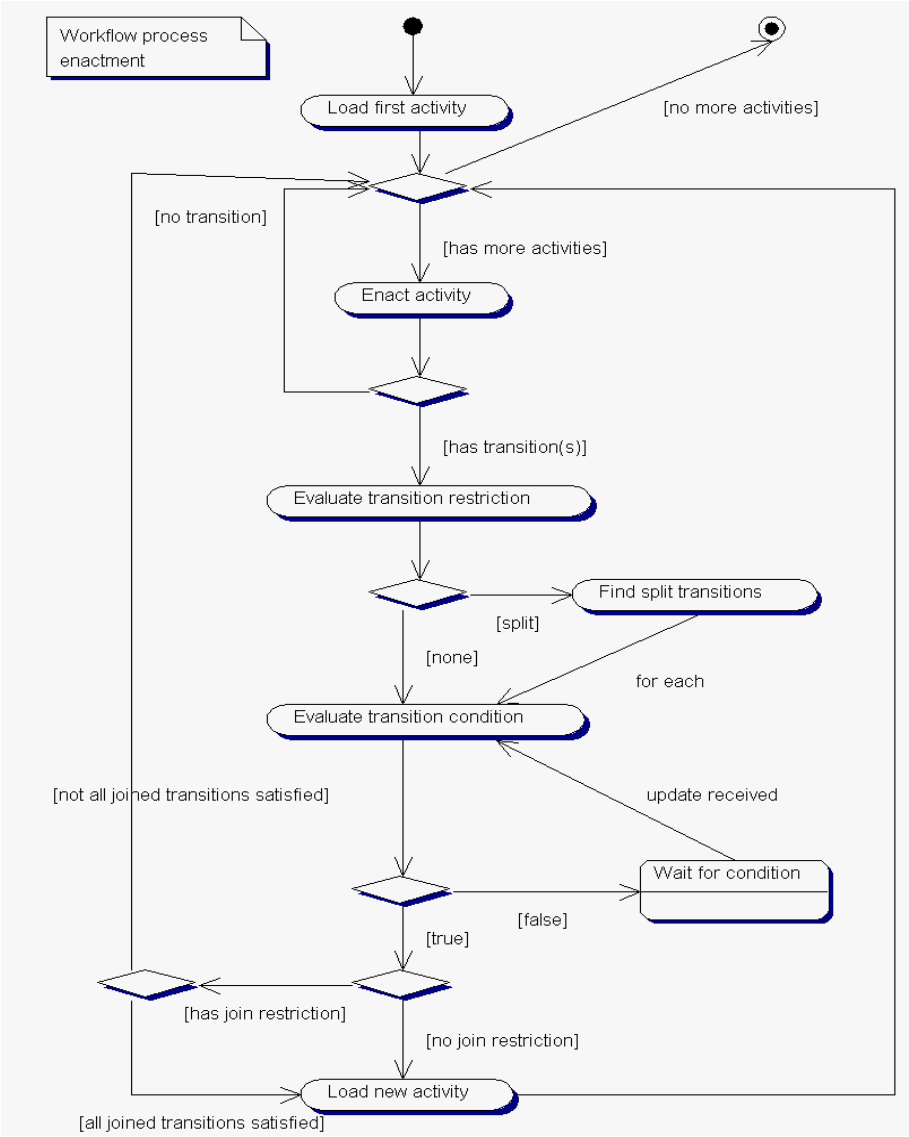

Figure 12.3: Workflow process enactment overview

Figure 12.3 shows the workflow process enactment performed by the *WorkflowProcessExecuter* class. The workflow process is started by loading the initial XPDL document from the file system. This is a simplification of requirement F23. After the workflow process is loaded, the workflow process executor enters the workflow enactment engine which is started by loading the first activity. The workflow enactment engine is a controller based on the observer pattern where we have a operating system thread responsible for managing the loading of activities, evaluation of transitions, and removal of finished activities. After the first activity is loaded, we enter the controller loop which will run as long as there are activities and transitions not yet executed. Activities are launched in separate operating system threads and these send a message to the observer when they have finished execution. When this happens, the observer will evaluate the transitions and transition restrictions of the activity. If it does not have a transition, that thread of execution will terminate. If the activity has a split transition restriction, we evaluate all the transitions in the split. If it has a transition, this is evaluated. When an activity is starting or has completed and when the workflow enactment status has changes, e.g., the workflow enactment is finished; the workflow enactment service is notified.

Transitions are evaluated by evaluating their condition. A transition without a condition always evaluates to true. Transition conditions evaluate to true if the specified data field has the specified value. If a transition evaluates to true, the activity the transition points to is loaded and enacted. If the transition evaluates to false, a transition exception is created and sent to the exception handler. The controller will then wait for the data field to be updated before evaluating the transition again. If the activity the transition points to has a join transition restriction, the activity will only be loaded when all transitions pointing to it has been satisfied.

This class implements requirements F1, F2, F3, F4, F5, and in cooperation with the *WorkflowEnactmentService* class requirements F6, F7, F8, F9, F10.

12.2.8 XPDLLoader Class

The XPDL loader is responsible for loading XPDL documents. It supports loading of both complete and incomplete XPDL documents. We have done our best to support the XPDL standard as it is, but some changes had to be made. The supported elements and changes are summarised in Table 12.1. Any elements not listed are not supported.

Supported XPDL elements	
FormalParameters	Fully supported.
Package	Attributes ignored.
Application	Description attribute not supported. ExtendedAttributes element not supported.
WorkflowProcess	ProcessHeader, RedefinableHeader, FormalParameters, DataTypes, ActivitySets, and ExtendedAttributes elements not supported. Only Id and Name attributes supported.
Activity	Limit, BlockActivity, StartMode, FinishMode, Priority, DeadLine, SimulationInformation, Documentation, and ExtendedAttributes elements not supported.
Implementation	SubFlow element not supported.

Tool	ExtendedAttributes not supported. Type loaded, but only APPLICATION type used in the workflow enactment.
ActualParameters	Modified the XPDL standard to support array values. This is considered to be a shortcoming of the current XPDL standard. See Appendix D.1 for the new XML schema for this element.
TransitionRestriction	Fully supported
Join	Fully supported, but only AND type used in the workflow enactment.
Split	Fully supported, but only AND type used in the workflow enactment.
TransitionRef	Fully supported.
Transition	ExtendedAttributes element not supported.
Condition	Only Type CONDITION supported.
Participant	ExternalReference and ExtendedAttributes elements type supported.
ParticipantType	Only Type SYSTEM and HUMAN supported.
DataField	ExtendedAttributes, Length, and Description elements not supported.
DataTypes	Only BasicType and ArrayType supported
BasicType	REFERENCE, DATETIME, and PERFORMER Type not supported.
ArrayType	Modified the XPDL standard to an array of BasicTypes. This modification should be considered a limitation if support for other data types than BasicType and ArrayType is added.

Table 12.1: Supported XPDL elements

This class implements requirements F1 and F2.

12.3 FWES EnactmentRepresentation Package

This package contains all classes representing elements in the workflow enactment representation. See Appendix B.3.1 for a class diagram. All elements are explained in detail in [55]. These classes together implement the domain model for requirements F1 and F2 and provide a basis for all other classes using the workflow process.

12.4 FWES Exception Package

The exception package contains the implementation of workflow exceptions, and implements requirement F9. Only the *TransitionException* class currently exist. This class extends the *WorkflowException* class and describes a transition exception. A transition exception occurs when a transition condition does not evaluate to true. The transition exception contains a reference to the transition that caused the exception.

12.5 FWES Scripting Package

The scripting package contains classes that simplify the usage of scripting in the application. These classes are tightly integrated with their respective scripting language.

The *LuaWorkflowProcess* class represents a workflow process in the Lua language. Currently it supports executing a transition condition query in the workflow process. It helps the *QueryExecuter* class in implementing requirement F4.

12.6 WCIS Package

We have in this component included some of the functionality of the CWIS component from the architecture in Chapter 11 in addition to the WCIS functionality.

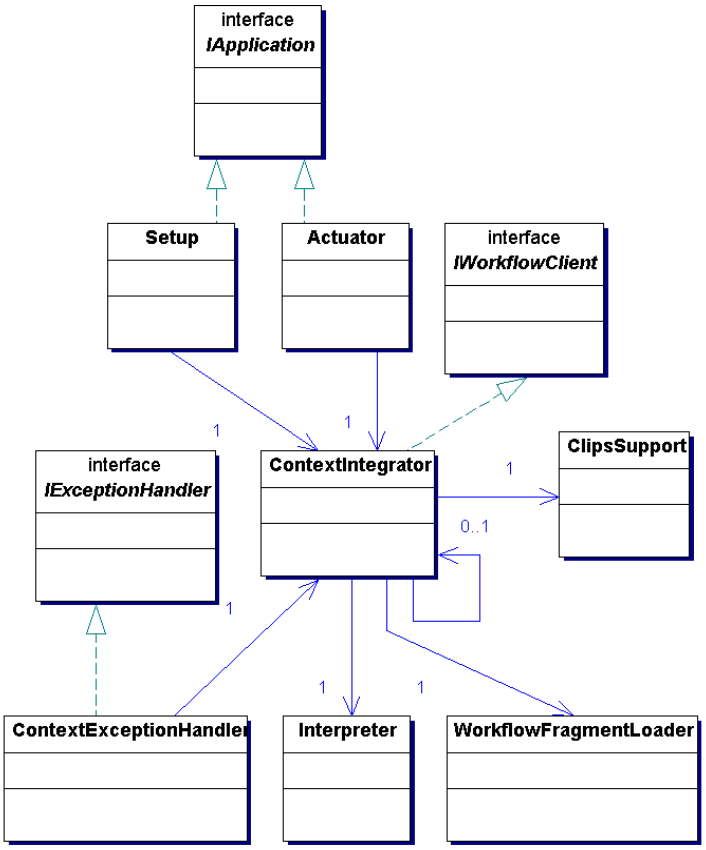

Figure 12.4: WCIS class diagram

12.6.1 ContextIntegrator Class

The *ContextIntegrator* class is responsible for managing context sources and context information. It is registered as a listener workflow client with the workflow enactment service, and will therefore receive all updates from the workflow enactment service. The context integrator is implemented with the singleton design pattern (Chapter 12.1.4). This enables

other classes, such as context-aware workflow applications and the exception handler, easy access to the context integrator instance.

The context integrator is capable of looking up sensors and actuators, handling transition exceptions, executing actuation commands, and receiving and interpreting context information from external actors. This context information is interpreted in the *Interpreter* class.

To be able to respond correctly according to rules, the context integrator uses a CLIPS (Chapter 7.2) expert system. When a change in the workflow enactment is notified to the context integrator, either from external sources or from the workflow enactment service, the CLIPS knowledge base is updated. If the facts in the knowledge base matches a rule, CLIPS triggers the correct method in the context integrator. Currently, the only supported method is "load new activity".

All work done in the context integrator is done in a separate thread so the application calling the context integrator can continue execution right after a call to the context integrator.

The *ContextIntegrator* class implements requirements F11, F12, F16, F17, F19, F22, F24, and F25. It also partially implements requirements F13, F14, and F15 by integrating it with the CORTEX Publish-Subscribe CF.

12.6.2 Interpreter Class

The interpreter implements a CORTEX FEL (Chapter 7.1.2) event parser. It is used as a component in the context integrator.

The FEL event must be on the form:

```
<subject>
    <Type>eventTypeName</Type>
    ...
</subject>
```

The *subject* tag name is replaced with the name of the FEL event subject. The *Type* tag content holds the event type for this event. All other tags are considered to be part of the event. Currently only the "updateSensor" event is supported. This event is on the form:

```
<subject>
    <Type>sensorUpdate</Type>
    <FieldName>xxx</FieldName>
    <Value>xxx</Value>
</subject>
```

The *FieldName* tag holds the sensor data field that has been updated and the *Value* tag holds the new value of that data field.

This class helps the *ContextIntegrator* class to integrate with the CORTEX Publish-Suscribe CF.

12.6.3 ContextExceptionHandler Class

When an exception occurs in the workflow enactment, the exception is sent to an exception handler. The context exception handler is a simple exception handler that just redirects the exception to the context integrator. Figure 12.5 shows how the exception handling of a transition exception works.

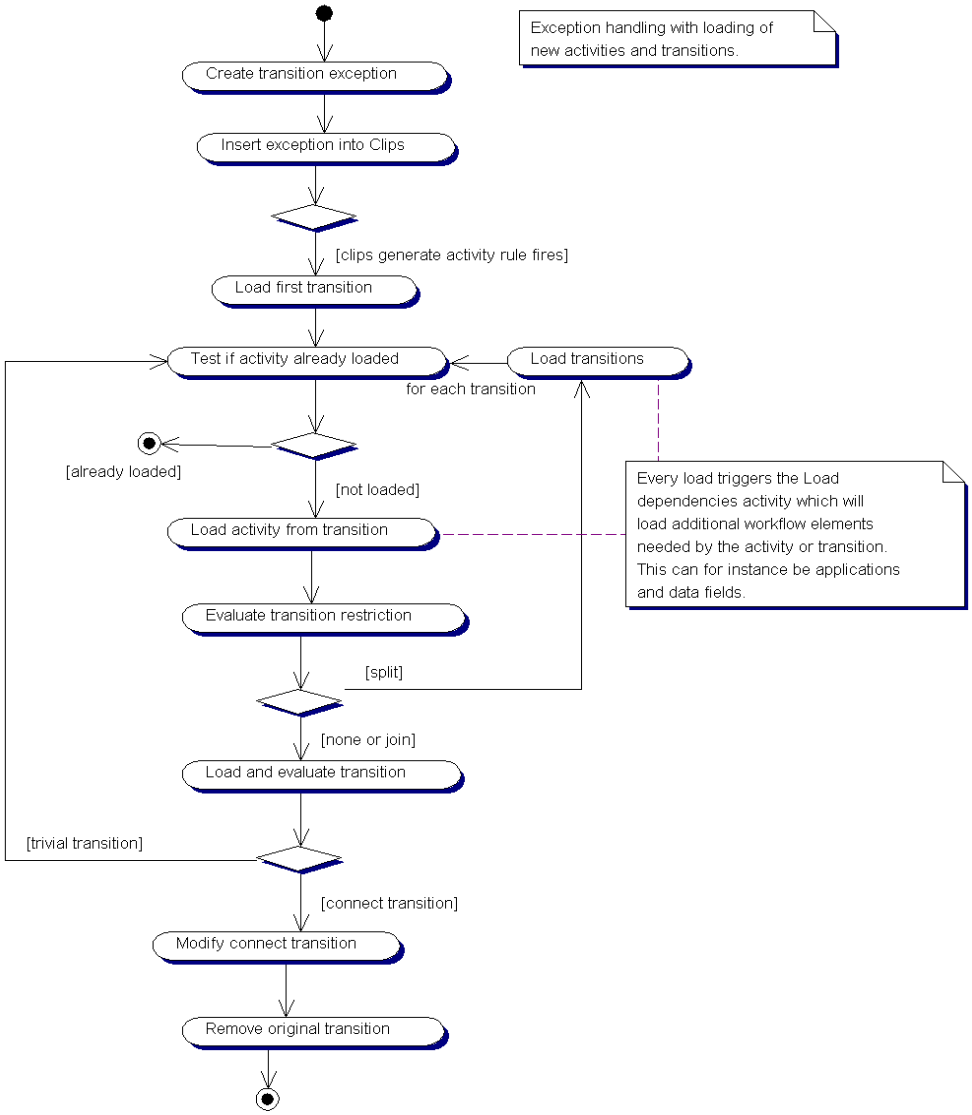

Figure 12.5: Exception handling

The workflow process executor detects a transition exception since a transition condition did not evaluate to true and sends a transition exception to the exception handler. Since this, in our case, is the context exception handler the transition exception is redirected to the context integrator. The context integrator inserts the fact that a transition exception has occurred into the CLIPS expert system. If an “add activity” CLIPS rule fires because

of this fact, the context integrator will add the required activities to the workflow process. The workflow enactment is paused, then the new activities are loaded by the workflow fragment loader into the workflow process, and finally the workflow process is resumed. Appendix H shows how an example workflow generates new actities from a transition exception.

This is an interface between the FWES component and the CWIS component when a workflow exception occurs. It is therefore a partial implementation of requirements F9 and F20.

12.6.4 WorkflowFragmentLoader Class

This class is responsible for loading workflow fragments from a XML file into the workflow process. The XML file must be of the form specified in the XML schema presented in Appendix C. Figure 12.5 shows how the loading of new activities occur. Currently only loading of transitions with activities and dependencies is supported.

The fragments have access to external information by using "?" as the value. External values are supported as described below:

- Transitions
  - First transition use the *From* value from the original transition. Current value is ignored.
  - Last transition use the *To* value from the original transition. Current value is ignored.
- Activities
  - May use the performer from the Activity found by using the *From* value from the original transition.
- Transitions
  - May use the condition from the original transition.

The *WorkflowFragmentLoader* class is a partial implementation of requirement F20.

12.7 WCIS Application Package

The application package contains workflow applications. Workflow applications are used by the workflow process during workflow enactment. These applications must be registered with the workflow enactment service before they can be used in the workflow enactment.

12.7.1 Setup Application

This application is responsible for setting up sensors and actuators for workflow processes and takes two arrays as input parameters: One array of sensors and one array of actuators to look-up and subscribe to. The setup application relies on the context integrator to do the actual work.

This is a direct implementation of requirement F16.

12.7.2 Actuator Application

The actuator application takes the actuator id and an actuator command as input parameters and uses the context integrator to execute the actuator command.

This is a direct implementation of requirement F17.

12.8 Utils Package

This package contains various utilities shared by the components in the workflow client.

It has a:

- *CLIPSSupport* class providing CLIPS support for the application. This class encapsulates common functionality to make CLIPS easier to use. See Chapter 7.2 for more information about CLIPS.
- *ScriptSupport* class providing access to the Lua scripting language. It provides a simplified interface for executing scripts and setting up the Lua scripting environment. See Chapter 7.3 for more information about Lua.
- *StringUtils* class since string conversion is often needed when developing a C++ application. This class simplify the common cases.

12.9 GUI

This is a simple GUI that provides an interface between the mobile worker and the workflow process enactment. It also provides a starting point for the workflow client application.

Figure 12.6 shows the elements in the GUI. An explanation of the elements is provided below:

Workflow Enactment Status This text field displays the current status of the workflow enactment. The workflow enactment can be in a started, paused, or stopped state.

Workflow Control Buttons These buttons control the workflow enactment. *Start* starts the workflow enactment, *Pause* pauses the workflow enactment, *R* resumes the workflow enactment, and *Stop* stops the workflow enactment.

Performer Name The name of the performer this workflow client (the GUI) represents. This is important because the workflow enactment service will only send activities to the GUI if the name is correct.

XPDL File Path The path to the XPDL file containing the workflow process to load when the workflow enactment is started.

Browse for XPDL File Button Opens the XPDL file selection dialogue.

Current Activity This text field shows information about the current registered activity. The mobile worker using the GUI is supposed to execute the activity manually and then push the *Activity completed* button.

Figure 12.6: GUI screenshot

Activity Completed Button The mobile worker can notify the workflow enactment service that the activity has been completed by pushing this button.

Data Field Id The Id of the data field to update.

Data Field Value The new value of the data field identified by the data field id text field. Pushing the *Set* button will set the new value.

Completed Activities List This is a list of activities completed in the workflow enactment service. Note that this includes all activities, including those the mobile worker has not performed.

Figure 12.7 shows the GUI when the workflow process enactment is running.

The GUI is an implementation of requirements F6 and F7.

12.10 Discussion

This design does not completely implement the architecture described in Chapter 11. We have done the simplifications described below.

- Data store is implemented as flat files stored on the device, and not in a data base.
- Cooperative workflow integration service is not implemented. Some parts of it is included in the workflow context integration service though.
- Server based workflow enactment service is replaced by a XPDL file stored on the device.

Figure 12.7: GUI active screenshot

- Re-validation of process paths is not included in the design because of time constraints.

Part VI

Implementation and Testing

Chapter 13

Implementation

This chapter gives an overview of the implementation by discussing relevant software metrics, project code structure, code samples, and encountered problems. The metrics are generated with the CCCC (Appendix A.1.2) tool.

13.1 Software Metrics

Software metrics help us understand the implementation of PocketFlow and provides measurements on issues related to object-oriented design and structural dependencies.

13.1.1 Summary of High Level Software Metrics

Table 13.1 presents an overview of high level software metrics for the implementation of PocketFlow.

Metric	**Value**
Number of Modules	53
Lines of Code	3636
Lines of Comment	702
LOC/COM	5.179

Table 13.1: Summary of software metrics

The "number of modules" metric is counted as the number of non-trivial modules (includes all classes, and any other module for which member functions are identified) and "lines of code" as the number of non-blank, non-commented lines of source code. The LOC/COM metric describes lines of comment pr. lines of code and is therefore a measurement on how well commented the code is.

13.1.2 Object Oriented Design

Here we discuss four of the six metrics proposed by Chidamber and Kemerer [9]. See Appendix I for a detailed presentation of these metrics per module on the included CD-

ROM. The four metrics are; weighted methods per class, depth of inheritance tree, number of children, and coupling between objects.

Weighted Methods per Class (WMC)

This metric is the sum of a weighting function over each module. Two versions of this metric are used. The first (WMC1) uses the nominal weight of 1 for each function. The second (WMCv) uses the nominal weight of 1 for each function accessible for other modules and 0 for private functions. Too high WMC1 and WMCv values for a module indicate that the module probably is too complex and should be decomposed into several less complex modules. All of our modules show a low value for WMC1 and low to moderate values for WMCv. Since the major part of these methods are getter and setter functions, we are confident we have good modularisation of our code.

Depth of Inheritance Tree (DIT)

This metric describes, for each module, the longest path of inheritance ending at that module. Our implementation has only 0 or 1 as values on modules for this metric, which indicates high possibility of easy reuse of the modules.

Number of Children (NOC)

This metric counts, for each module, how many modules that inherit from it. Our implementation has moderate values for this metric which again indicates easy reuse of modules.

Coupling Between Objects (CBO)

The CBO metric counts, for each module, the number of other modules that are coupled to it. High coupling indicates a lack of module encapsulation. For our implementation, the *ContextIntegrator*, *WorkflowProcess*, and *WorkflowProcessExecuter* modules exhibit moderate CBO values, and the rest of the modules low CBO values. Since the *WorkflowProcess* modules works as an interface between a large number of modules, and since *WorkflowProcessExecuter* and *ContextIntegrator* is the main modules in the workflow client module, we believe our implementation can be considered to have loose coupling.

13.1.3 Structural Metrics

Here we discuss the fan-in, fan-out, and information flow measure metrics for our implementation. The fan-in metric represents, for each module, the number of modules which pass information into it. Fan-out represents, for each module, the number of other modules for which it passes information. The information flow metric is a metric for structural complexity which is calculated as the square of the product of the fan-in and fan-out of a single module.

Most of our modules show low fan-in and fan-out values, except the *WorkflowEnactmentService*, *WorkflowProcessExecuter*, and *WorkflowProcess*. This is expected since these modules also have a larger number of couplings to other modules.

13.2 Code Samples

In this section we will demonstrate source code to make statements presented previously in the report more understandable. Note that this is only a short introduction to the PocketFlow source code and the reader should refer to the source code on the CD-ROM (Appendix I) to get a complete understanding.

13.2.1 Starting a Context-Aware Workflow Process

Listing 13.1 shows the instantiation of a workflow enactment service capable of running a context-aware workflow process.

Listing 13.1: Starting a context-aware workflow process

```
1  WorkflowEnactmentService workflowEnactmentService = new WorkflowEnactmentService();
2  workflowEnactmentService->loadWorkflowProcess(CONTEXT_AWARE_WORKFLOW);
3  workflowEnactmentService->registerWorkflowClient(this); // in this case the GUI class
4  workflowEnactmentService->registerWorkflowClient(ContextIntegrator::getInstance());
5  workflowEnactmentService->registerApplication(new Setup());
6  workflowEnactmentService->registerApplication(new Actuator());
7  workflowEnactmentService->setExceptionHandler(new ContextExceptionHandler());
8
9  ContextIntegrator::getInstance()->start(KNOWLEDGE_BASE_FILE,
       WORKFLOW_FRAGMENTS_FILE);
10
11 workflowEnactmentService->startWorkflowProcess();
```

In this example we run a context-aware workflow process with two context-aware workflow applications: *Setup* and *Actuator*, the context integrator as a workflow client, and the GUI as a workflow client. We also use a context-aware exception handler.

The constants in Listing 13.1 are as follows:

- CONTEXT_AWARE_WORKFLOW: The path to the XPDL file containing the context-aware workflow process.
- KNOWLEDGE_BASE_FILE: The path to the CLIPS knowledge base.
- WORKFLOW_FRAGMENTS_FILE: The path to the workflow fragments file.

Note that the context integrator is started before the workflow process since the workflow process may need to setup actuators and sensors when starting. This is a requirement of this prototype. Context integrator processes should also be able to run autonomically and provide services to workflow processes.

This is an implementation of the dependency injection pattern described in Chapter 12.1.2.

13.2.2 Generating New Activities

The *ContextIntegrator::generateNewActivity* method shown in Listing 13.2 is executed (the ASSERT macro is explained in Chapter 14.6.1) when the CLIPS expert system derives the need for new activities.

Listing 13.2: Generating new activities

```
1  void ContextIntegrator::generateNewActivity(CString transitionFragmentId, CString
       originalTransitionId) {
2      TRACE(_T("ContextIntegrator::generateNewActivity()"));
3      FWES::EnactmentRepresentation::WorkflowProcess* workflowProcess = this->
           workflowClient_workflowEnactmentService->getWorkflowProcess();
4      ASSERT(workflowProcess != NULL);
5      Transition* originalTransition = workflowProcess->getTransitionById(originalTransitionId);
6      ASSERT(originalTransition != NULL);
7      CString from = CString(originalTransition->getFrom());
8
9      this->workflowClient_workflowEnactmentService->pauseWorkflowProcess();
10     this->workflowFragmentLoader->loadActivities(transitionFragmentId, originalTransition,
           workflowProcess);
11     Transition* newTransition = workflowProcess->getTransitionByFrom(from);
12     ASSERT(newTransition != NULL);
13
14     this->workflowClient_workflowEnactmentService->updateTransition(originalTransition,
           newTransition);
15     this->workflowClient_workflowEnactmentService->resumeWorkflowProcess();
16 }
```

First the original transition, (the transition that caused the generation of new activities), is loaded from the workflow process. Then the workflow process enactment is paused, before new activities are loaded into the workflow process. Finally, the workflow process enactment is resumed.

13.2.3 Asserting Transition Exceptions

Listing 13.3 shows how transition exceptions are asserted into the CLIPS expert system. The transition exception is asserted with it's identifier (Id) to make it possible to get the reference to the transition at a later time (e.g. when generating new activities).

Listing 13.3: Asserting transition exceptions

```
1  void ContextIntegrator::handleTransitionException(FWES::exception::TransitionException*
       transitionException) {
2      TRACE(_T("ContextIntegrator::handleTransitionException()"));
3      ASSERT(transitionException != NULL);
4      ASSERT(transitionException->getTransition() != NULL);
5      this->clipsSupport->assertString(L"(transitionException␣\"" + transitionException->
           getTransition()->getId() + "\")");
6      this->clipsSupport->run(CLIPS_MAX_RUN_TIME);
7  }
```

13.2.4 Evaluate Activity Transition

The *WorkflowProcessExecuter::evaluateActivityTransition* method shown in Listing 13.4 is responsible for evaluating any transition(s) an activity may have.

Listing 13.4: Evaluate activity transition

```
1  void WorkflowProcessExecuter::evaluateActivityTransition(Activity* activity) {
2      PROCESS_EXECUTER_TRACE(_T("WorkflowProcessExecuter::evaluateActivityTransition
           ();␣Activity␣Id:␣%s"), activity->getId());
3      // check if the activity has some transition restrictions
4      bool hasSplit = false;
5      if (!(activity->getTransitionRestrictions().empty())) {
6          PROCESS_EXECUTER_TRACE(_T("WorkflowProcessExecuter::
               evaluateActivityTransition();␣Has␣transition␣restrictions"));
7          std::vector<TransitionRestriction*> restrictions = activity->getTransitionRestrictions
               ();
8          std::vector<TransitionRestriction*>::iterator currentRestriction;
9          for (currentRestriction = restrictions.begin(); currentRestriction != restrictions.end();
               currentRestriction++) {
10             if ((*currentRestriction)->isSplit()) {
11                 evaluateSplitRestriction((*currentRestriction)->getSplit());
12                 hasSplit = true;
13             }
14         }
15     }
16     else {
17         PROCESS_EXECUTER_TRACE(_T("WorkflowProcessExecuter::
               evaluateActivityTransition();␣No␣transition␣restrictions"));
18     }
19     // ok no transitions restrictions, do we have a transition?
20     Transition* transition = this->workflowProcess->getTransitionByFrom(activity->getId())
           ;
21     if (!hasSplit && transition != NULL) {
22         evaluateTransition(transition);
23     }
24     else {
25         // nope, this is the end of this thread of execution
26         PROCESS_EXECUTER_TRACE(_T("WorkflowProcessExecuter::
               evaluateActivityTransition();␣No␣transition"));
27     }
28     removeRunningActivity(activity);
29 }
```

First the activity is checked for any transition restrictions. If it has one or more transition restrictions, all split transition restrictions are evaluated in the *WorkflowProcessExecuter::evaluateSplitRestriction* method shown in Listing 13.5.

Finally, if the activity has a transition and no split transition restrictions were detected earlier, the transition is evaluated in the *WorkflowProcessExecuter::evaluateTransition* method shown in Listing 13.6.

Listing 13.5: Evaluate split transition restriction

```
1  void WorkflowProcessExecuter::evaluateSplitRestriction(Split* split) {
2      PROCESS_EXECUTER_TRACE(_T("WorkflowProcessExecuter::evaluateSplitRestriction()")
           );
3      std::vector<TransitionRef*> transitionRefs = split->getTransitionRefs();
```

```
4     std::vector<TransitionRef*>::iterator currentRef;
5     for (currentRef = transitionRefs.begin(); currentRef != transitionRefs.end(); currentRef++)
          {
6         TransitionRef* theTransitionRef = *currentRef;
7         // get the real transition
8         Transition* theTransition = this->workflowProcess->getTransitionById(
              theTransitionRef->getId());
9         if (theTransition != NULL) {
10            evaluateTransition(theTransition);
11        }
12        else {
13            STLOG_WRITE(_T("WorkflowProcessExecuter::evaluateSplitRestriction(); Error:
                  Was going to evaluate the transition with id %s but it was not found!"),
                  theTransitionRef->getId());
14        }
15    }
16 }
```

The *WorkflowProcessExecuter::evaluateSplitRestriction* method shown in Listing 13.5 loads the transition for each of the transition references in the split transition restriction and evaluates it in the *WorkflowProcessExecuter::evaluateTransition* method shown in Listing 13.6

Listing 13.6: Evaluate transition

```
1  void WorkflowProcessExecuter::evaluateTransition(Transition* transition) {
2      if (transition == NULL) {
3          PROCESS_EXECUTER_TRACE(_T("WorkflowProcessExecuter::evaluateTransition();
               Error! The transitions was NULL!"));
4          return;
5      }
6
7      PROCESS_EXECUTER_TRACE(_T("WorkflowProcessExecuter::evaluateTransition();
           Transitions id=%s, from=%s, to=%s"), transition->getId(), transition->getFrom(),
           transition->getTo());
8      if (transition->isSatisfied() == false) {
9          if ( transition->getCondition() != NULL) {
10             Condition* condition = transition->getCondition();
11             bool satisfied = this->queryExecuter->evaluateCondition(condition->
                   getXpression());
12             if (satisfied) {
13                 PROCESS_EXECUTER_TRACE(_T("WorkflowProcessExecuter::
                       evaluateTransition(); Transition condition satisfied."));
14                 transition->setSatisfied(true);
15             }
16             else {
17                 PROCESS_EXECUTER_TRACE(_T("WorkflowProcessExecuter::
                       evaluateTransition(); Transition condition not satisfied."));
18                 this->waitingTransitions.push_back(transition);
19                 // notify the exception handler
20                 PostThreadMessage(this->executerThreadId,
                       THRD_MESSAGE_NOTIFY_TRANSITION_EXCEPTION, (WPARAM)
                       transition , 0);
```

```
21                  return; // must wait for the condition to be satisfied
22              }
23          }
24          else {
25              // transition without condition is always true
26              PROCESS_EXECUTER_TRACE(_T("WorkflowProcessExecuter::evaluateTransition
                    ();␣No␣condition,␣setting␣satisfied␣=␣true"));
27              transition->setSatisfied(true);
28          }
29      }
30      // the condition must be satisfied, create the new activity
31      Activity* nextActivity = this->workflowProcess->getActivityById(transition->getTo());
32      if (nextActivity != NULL) {
33          bool create = true;
34          if (nextActivity->hasJoinRestriction()) {
35              PROCESS_EXECUTER_TRACE(_T("WorkflowProcessExecuter::evaluateTransition
                    ();␣Has␣join␣transitions␣restriction"));
36              // this must be joined
37              create = evaluateJoinTransition(transition);
38          }
39          if (create == true) {
40              HANDLE nextActivityThread = addRunningActivity(nextActivity);
41          }
42          else {
43              PROCESS_EXECUTER_TRACE(_T("WorkflowProcessExecuter::evaluateTransition
                    ();␣Transition␣with␣id␣%s␣must␣wait␣for␣join"), transition->getId());
44          }
45      }
46      else {
47          STLOG_WRITE(_T("WorkflowProcessExecuter::evaluateTransition();␣Error:␣Should␣
                have␣created␣a␣new␣activity␣with␣id␣%s␣but␣it␣was␣not␣found!"), transition->
                getTo());
48      }
49  }
```

Transitions are evaluated by evaluating their condition as shown in Listing 13.6. A transition without a condition always evaluates to true. Transition conditions evaluate to true if the specified data field has the specified value. If a transition evaluates to true the activity the transition points to is loaded and enacted. If the transition evaluates to false, a transition exception is created and sent to the exception handler. The controller will then wait for the data field to be updated before evaluating the transition again. If the activity the transition points to has a join transition restriction, the activity will only be loaded when all transitions pointing to it have been satisfied as shown in Listing 13.7.

Listing 13.7: Evaluate join transition

```
1  bool WorkflowProcessExecuter::evaluateJoinTransition(Transition* transition) {
2      if (transition == NULL) {
3          PROCESS_EXECUTER_TRACE(_T("WorkflowProcessExecuter::evaluateJoinTransition
                ();␣Error!␣The␣transition␣was␣NULL!"));
4          return false;
5      }
```

```
6     PROCESS_EXECUTER_TRACE(_T("WorkflowProcessExecuter::evaluateJoinTransition();␣
          Transition␣id=%s,␣from=%s,␣to=%s"), transition->getId(), transition->getFrom(),
          transition->getTo());
7     std::vector<Transition*> joinTransitions = this->workflowProcess->getTransitionsByTo(
          transition->getTo());
8     std::vector<Transition*>::iterator currentTransition;
9     for (currentTransition = joinTransitions.begin(); currentTransition != joinTransitions.end();
          currentTransition++) {
10        if ((*currentTransition)->isSatisfied() == false) {
11            return false; // not all join transitions have been satisfied
12        }
13    }
14    return true; // all joined transitions are now satisfied
15 }
```

13.2.5 Actuator Workflow Application Execution

Listing 13.8 shows an example of how a workflow application may connect to context sources. In this case it is an actuation order to an actuator. Firstly the actuator retrieves its parameters, then the actuator command is sent to the context integrator.

Listing 13.8: Actuator workflow application execution

```
1  void Actuator::application_execute(FWES::EnactmentRepresentation::WorkflowProcess*
       workflowProcess, std::vector<FWES::EnactmentRepresentation::ActualParameter*>
       actualParameters) {
2     TRACE(L"Actuator::application_execute()");
3     ASSERT(workflowProcess != NULL);
4     ASSERT(actualParameters.size() == 2);
5     ASSERT(actualParameters[0]->getBasicValue() != NULL);
6     ASSERT(actualParameters[1]->getBasicValue() != NULL);
7
8     CString actuatorName, actuatorCommand;
9     if (actualParameters[0]->getIndex() == 1) {
10        actuatorName = actualParameters[0]->getBasicValue()->getStringValue();
11        actuatorCommand = actualParameters[1]->getBasicValue()->getStringValue();
12    }
13    else {
14        actuatorName = actualParameters[1]->getBasicValue()->getStringValue();
15        actuatorCommand = actualParameters[0]->getBasicValue()->getStringValue();
16    }
17    contextIntegrator->executeActuatorCommand(actuatorName, actuatorCommand);
18 }
```

13.3 Project Structure

The project is divided into three static libraries[1], and two executables implementing the libraries.

[1] A library statically linked into an executable at compile-time, as apposed to dynamic link libraries which are linked at run-time.

WorkflowClient

This is a static library containing the workflow client described in Chapter 12.

Clips

This project constains the CLIPS expert system static library.

Luacelib

The luacelib static library contains the Lua language execution engine.

GUI

The GUI project contains the GUI executable. It links to the WorkflowClient, Clips, and Luacelib static libraries.

Unit

The Unit project contains the unit test suite executable. We will discuss unit testing in the nest chapter. The unit executable links to the WorkflowClient, Clips, and Luacelib static libraries.

13.4 Encountered Problems

We encountered problems with unstable network connectivity during the integration of the CORTEX middleware into our implementation. The CORTEX middleware is developed and tested using IPAQ PDAs. The Fujitsu Siemens PDAs we used are very unstable when used in ad-hoc wireless network mode. We decided to simulate how the PDA receives events by loading events from local storage instead. Even though this problem prohibited us from testing our implementation on several PDAs over a wireless network, it does not have a major effect on testing of our design and discussion of our research questions.

Chapter 14

Testing

This chapter presents how we conducted testing, which testing framework we used, and descriptions of the tests we performed.

14.1 Testing Strategy

Since our development is based on the XP methodology, we have written and run tests during implementation in a incremental manner. All unit tests have been run on a Fujitsu Siemens PDA, and not on the emulator. The steps carried out when testing a module were:

1. Copy test resources to the device (XPDL, etc.).
2. Compile and link project files in Embedded Visual Studio.
3. Deploy unit.exe to the device.
4. Run the unit tests on the device and receive feedback on test outcome.
5. View generated log.

14.2 Use of Logging

We have during the development of PocketUnit made heavy use of logging. In PocketFlow we have two types of logging; trace and regular log. Trace logs will only be executed when the application is built with the *DEBUG* flag. Regular logs are always executed regardless of if the *DEBUG* flag is set or not. The log is written to a file with the name "<executable name>_log.txt".

The generated log file has been an important help in detecting and fixing errors quickly, especially when executing more than one operating system threads.

14.3 Unit Testing with PocketUnit

All unit tests are run with the PocketUnit test framework. PocketUnit along with test-classes are included on the accompanying CD-ROM (Appendix I). The rest of this chapter presents unit tests for the PocketFlow implementation.

14.3.1 FWES

The FWES package described in Chapter 12.2 is tested with the following unit test cases.

TestXPDLLoader Test Case

The purpose of this unit test is to test the XPDL-loader for loading of XPDL-documents from local storage on the device. A workflow process with associated applications, activities, participants, and data fields are loaded. The test checks if the elements are loaded correctly, have the right types, and not a *NULL* value.

TestWorkflowProcessExecuter Test Case

The purpose of the *TestWorkflowProcessExecuter* unit test is to test the execution of a workflow process, and check if transitions trigger between activities based on data field values. The test creates a workflow process, sets up a workflow enactment service, and starts the workflow process executor.

TestQueryExecuter Test Case

The *TestQueryExecuter* unit test is responsible for testing the evaluation of transition conditions. A transition workflow element is loaded from XPDL with conditions. Expressions are retrieved from conditions and evaluated against data field values.

TestEnactmentRepresentation Test Case

The purpose of this unit test is to test the representation of workflow process elements for use in enactment. For instance, test if data fields are set to the correct data types with the correct values.

14.3.2 WCIS

The WCIS package is described in Chapter 12.6 and is tested with the following unit test cases.

TestWorkflowFragmentLoader Test Case

The purpose of the *TestWorkflowFragmentLoader* test case is to test loading of workflow fragments from local storage on the device. After the new fragments have been loaded, the workflow process is tested if it contains all the new elements.

TestContextAwareWorkflow

This test runs the simple context aware workflow process show in Appendix H. After the workflow process is run it is checked for new elements.

14.3.3 Utils

The utils package is described in Chapter 12.8 and is tested with the following unit test cases.

TestClips Test Case

The *TestClips* unit test is responsible for testing the CLIPS integration with the WCIS component. The test includes loading a knowledge base from file, defining external C functions, asserting facts in knowledge base, and running the expert system.

TestScriptExecuter Test Case

This test case tests that the script support executes condition evaluations correctly.

14.4 Manual Testing

While we try to do automated testing as much as possible there are situations where this is not feasible. For instance when testing the GUI widgets, it is often necessary to sift through the log file to find where an error occurred.

14.5 Not Tested

The code coverage of our tests is far from 100%, and refactoring of the code should be done with great care. In addition we have no tests for external code like the CORTEX middleware, CLIPS, and Lua.

14.6 Debugging

When programming in any language, and especially in C++ since it is not protected by a managed virtual machine like Java and C#, there will occur unknown errors often setting the application in an unrunnable state. To avoid this we have used the *ASSERT* macro as well as the debugging facilities in MS Embedded Visual Studio.

14.6.1 The ASSERT Macro

The *ASSERT* macro in C++ helps us identify potentially dangerous situations that may lead to a fatal error (a crash). For instance, by using the *ASSERT* macro we can check if a variable that never should have the NULL value actually does not have the *NULL* value like this: *"ASSERT(variable != NULL)"*. If the variable is in fact *NULL*, an error dialogue containing the file and the line where the assertion error occurred is shown on the device. It is then usually trivial to find the error. When the application is built without the *DEBUG* flag the *ASSERT* macro is removed from the executable. The *ASSERT* macro is therefore non-intrusive.

14.6.2 Debugging in Embedded Visual Studio

The MS Embedded Visual Studio debugging environment provides us with an excellent platform for finding unexpected errors in the application. It is especially useful for finding fatal errors that cause the application to crash since it gives us the stack trace to the fatal error.

Part VII

Discussion and Conclusion

Chapter 15

Discussion

In this chapter we evaluate the design and implementation of PocketFlow and provide a discussion on supported requirements, what we have achieved, and which aspects remain unresolved. We also evaluate the research questions and how our research methods helped us in our work.

15.1 Evaluation of PocketFlow

Not all requirements have been implemented in the PocketFlow prototype. Here we discuss the supported requirements and how the workflow enactment and context-awareness are implemented.

15.1.1 Supported Requirements

We present a summary of which requirements that are supported by PocketFlow, since we have defined several requirements outside the scope of our work. A requirement can be supported by the architecture, the design, the implementation, or not at all. Any requirement supported by the implementation is also supported by the design, and any requirement supported by the design is also supported by the architecture.

Table 15.1 shows the supported requirements for a basic workflow system. PocketFlow supports all requirements except F3 which is only supported by the design, some requirements are only partly supported. The workflow enactment service not completely functional. Enactment of block and sub-process activity type are for instance not supported. The GUI implementation is also rather basic.

Id	Description	Supported
F1	Adhere to the WfMC interface 1 meta-model. Support a subset of the XPDL process definition language.	Implementation
F2	Interpret and enact a subset of the process definitions defined in the XPDL.	Implementation
F3	Use activities completed by other workflow participants as pre-condition for own activities.	Design

F4	Evaluate conditions for transitions between activities by checking data fields in the workflow process.	Implementation
F5	Perform enactment of a single workflow process or workflow process fragment.	Implementation
F6	Provide workflow enactment feedback and information to the mobile worker.	Implementation
F7	Support receiving feedback and information from the mobile worker.	Implementation
F8	Communicate with other workflow clients in order to send and receive activities.	Implementation
F9	Exception handling of transitions that does not have a satisfied condition.	Implementation
F10	A workflow process must be able to execute external workflow applications.	Implementation

Table 15.1: Supported basic workflow system requirements

While PocketFlow has support for all context-aware workflow system requirements in the design, not all requirements are supported by the implementation as shown in Table 15.2. This is the result of not being able to implement the CORTEX Publish-Subscribe CF middleware in PocketFlow because of the networking problems described in Chapter 13.4.

Id	**Description**	**Supported**
F11	Interpret and enact process definitions needed for context-aware workflows. Context information should be used in the evaluation of workflow transitions (pre-condition for activities).	Implementation
F12	Ad-hoc start of processes and activities based on context information.	Design
F13	Filter sensor data, based on rules, to provide data the context client can use directly.	Design
F14	Service for the discovery and look-up of context-sources.	Design
F15	Support both polling and publish / subscribe mechanisms for context information retrieval from context-sources, sensors, and actuators.	Design
F16	Support the definition of context-source look-up, polling, and conversion of the context information between the context representation and the workflow representation as XPDL elements.	Implementation
F17	Support the sending of actuation orders to actuators.	Implementation

Table 15.2: Supported context-aware workflow system requirements

We believe PocketFlow is a good starting point to develop adaptive workflow systems, but there are several unresolved issues in our implementation as shown in Table 15.3. This is again mostly due to our inability to get the CORTEX Publish-Subscribe CF middle-

ware to work with the unreliable network connection, but also because our focus when implementing the prototype was not on cooperation between several mobile workers.

Id	Description	Supported
F18	Perform distributed enactment of concurrent processes over several workflow clients that possibly competes for resources.	Architecture
F19	Support template-based workflow enactment.	Implementation
F20	Perform situated planning by supporting exception handling of undefined contextual states by creating new activities based on a set of rules, workflow fragments, context-sources, and/or manual interaction.	Implementation
F21	Support manual interaction from the mobile worker to create, modify, or remove activities.	Architecture
F22	Provide persistent storage of rules, workflow fragments and templates, workflow history and documentation for the user for easy retrieval.	Architecture
F23	Support central storage of workflow templates.	Architecture
F24	Must be able to receive activities from other participants of the distributed workflow process.	Design
F25	Must be able to coordinate with other workflow clients with conflicting interest to decide, based on predefined and ad-hoc rules, which workflow client gets priority.	Architecture
F26	Support re-validation of selected process paths, if the current path does not lead to the process goal.	Architecture

Table 15.3: Supported adaptive ad-hoc workflow system requirements

As shown in Table 15.4, PocketFlow does not support any of the mobility requirements explicitly. Some of these requirements may be supported by the CORTEX Publish-Subscribe CF middleware, but this has not been tested.

Id	Description	Supported
F27	Support for physical mobility and network mobility.	Not supported
F28	Handling of unreliable communication.	Not supported
F29	Support for disconnected operations and asynchronous communication.	Not supported
F30	Support for session mobility.	Not supported

Table 15.4: Supported mobility requirements

15.1.2 Evaluation of the Workflow Enactment

The workflow enactment support in PocketFlow should be considered simplistic since it does not support important constructs such as sub-flows, but we strongly believe it is easily extended to support a more complete workflow enactment. PocketFlow should also

handle more workflow enactment exception types when, for instance, an enactment element is missing or has an invalid value.

15.1.3 Evaluation of the Adaptive Workflow Implementation

Our implementation of adaptive workflow is based on an expert system that is responsible for modifying the workflow according to facts and rules. The local workflow enactment service and external entities like context-sources and other workflow clients update the fact data store on the expert system, and the expert system triggers relevant functions if its rules match the facts. While this arguably creates an adaptive and ad-hoc workflow, it does have a few shortcomings. The rules in the data store are not shared between workflow participants, and there is no learning involved (i.e. no new rules are created at run-time). We believe this is a serious shortcoming of the current implementation, but it should not be too hard to implement this in a future version of PocketFlow.

15.1.4 Evaluation of the Context Implementation

We believe the current CORTEX Publish-Subscribe CF is useful for look-up of context-sources, and for handling communication, but because of the WLAN problems on our devices we cannot be certain.

Furthermore, we believe our implementation of context in the workflow process execution is non-intrusive with the WfMC reference model since we do not use any non-standard constructs. Theoretically, it should be possible to use the same architecture for implementation of context in any standard-compliant workflow enactment service.

15.2 Evaluation of the Research Questions

Both the literature study and prototype development helped us reach partial answers to our research questions. We summarise our findings below.

15.2.1 RQ1

How can a workflow process adapt to environmental changes, and how can a workflow process modify the environment?

This research question is divided into two parts; we will discuss each part separately.

How can a workflow process adapt to environmental changes? A workflow process can use context as pre-conditions for activities. This is solved by using data fields in the workflow process as context representation. Transitions in the workflow enactment may then use these data fields as transition conditions.

When a transition is not satisfied a transition exception is created, and based on the expert system rules and facts, new activities may or may not be created.

It is also possible that an environmental change happens outside the context-sources known by the workflow process. The expert system will intercept these changes and based on the expert system rules and facts, new activities may or may not be created.

This approach works as long as rules can be defined for all possible environmental states. In the real world that is not possible since there is too many changing environmental states. We therefore propose to expand our approach with a distributed knowledge base with learning capabilities, as proposed by the intelligent artifacts approach.

How can a workflow process modify the environment? The workflow process can modify the environment by using actuators and sending activities to other workflow clients. If we consider actuators to be any software or hardware unit in the environment that can execute commands, it should be possible to modify the entire environment. Actuations are sent either from the workflow process itself or from the expert system. Activities are exclusively sent by the expert system.

While we have not looked into security aspects of modifying the environment, we believe this approach is feasible.

15.2.2 RQ2

How can we achieve automatic definition of activities based on process goals and environmental context?

We use an expert system to decide when and how new activities are created. The expert system use rules and facts to infer the correct action. Activities are created by loading and instantiating workflow fragments, where workflow fragments are parts of a workflow process that may be used in several workflow processes. When the workflow fragments are loaded into the workflow process, the workflow process can be considered to be a template since the workflow process has not connected to any context-sources. After the workflow template has connected to its context-sources, the workflow process is fully functional.

Again this approach is heavily dependent on good rules in the expert system. As mentioned in RQ1, we need a distributed knowledge base capable of learning new rules.

15.2.3 RQ3

How can we achieve ad-hoc workflow enactment in a mobile computing environment?

The solutions for RQ1 part one and RQ2 also applies to RQ3. We need a distributed rule base with several workflow clients performing cooperative reasoning. This reasoning could be based on virtual organisation theory (see [27] for a more detailed discussion). By using workflow fragments, again distributed among the workflow clients, we can achieve ad-hoc workflow definition and hence ad-hoc workflow enactment.

15.3 Evaluation of Research Method

We used an engineering approach to research when trying to develop solutions for the problem definition from Chapter 1.3. This approach has included scenario building with requirements elicitation and prototype development with use of elements from the extreme programming methodology.

Literature Survey

The literature survey has provided us with an understanding of relevant topics, which has proven useful when discussing aspects of our architecture and design.

Scenario Building and Requirements Elicitation

Our field study at Aker Verdal and the related scenario building were useful as a showcase for an environment that can benefit from the kind of future technology we envision in our work. This helped us understand the requirements needed for a system supporting adaptive mobile work.

Extreme Programming

Our use of the XP-methodology worked well when developing the PocketFlow prototype. Throughout the implementation we defined unit tests which are described in Chapter 14.3. The requirements and design were incrementally improved in a evolutionary manner, based on our increased understanding of the problem domain. We used code review to discover programming errors quickly. Other practices such as pair programming were deemed too risky since we did not have any experience with them, and because of the scope and type of this project. The project was partly exploratory, investigating new technological possibilities and thus we had to split our attention to different parts of the architecture in the implementation phase. In that respect, we have not performed a XP project by the book, but adopted the practises most useful for our project.

Chapter 16

Further Work

This chapter focus on issues we deem to be important, but have not had time to address.

16.1 Prototype Implementation

Support for more XPDL elements

The enactment representation package (12.3) and the workflow process executor (12.2.7) needs to be extended with support for additional XPDL elements for the prototype to perform more advanced workflow enactment. The elements which needs to be supported includes subflows, XOR split and join, and more transition condition types.

Support for advanced fragment loading

Chapter 3.6.3 provides state-of-the-art on creating workflow processes on the fly. Design and development of a more dynamic fragment loading should be considered as an important part of any further work. With the current implementation, it is only possible to load transitions which in turn load activities and other dependencies.

Advanced GUI

PocketFlow GUI for PDAs is described in Chapter 12.9. This GUI needs to be extended to provide the mobile worker with more information on the workflow enactment. Additional information can be images, documentation, etc., which provide an increased understanding of the state of the environment, and increased support for the work to be done.

SQL database

Workflow clients may in a future implementation use an SQL database for persistent storage of information related to the contextual state of the environment. This information can then be used to provide workflow enactment and sensor data history. It can also be used to enable learning in the workflow definition process, creating activity templates, etc.

Use WfMC interface 2 and 3

Future work may consider using WfMC interface 2 and 3 to standarise interoperability with other workflow systems. This may help integrate PocketFlow with other standard compliant systems.

Increased scripting support

Lua scripts allow advanced queries and are used by the query executor. Increased scripting support would make the prototype more flexible since more parts of the code can be modified without recompiling the application.

Support for more types of workflow enactment exceptions

When enacting a workflow process, we only throw an exception whenever there is a transition exception. Ideally, we should throw an exception for any anomaly in the workflow process execution. For instance, when workflow applications or other workflow elements can not be found, etc.

Support for additional context event types

The context integrators interpreter (12.6.2) supports interpretation of sensor update events. Logic for interpreting received activities and workflow fragments is not implemented.

Support for SensorML

Chapter 6.3 introduces SensorML, a standard for communication between workflow clients and sensors. Future work should consider the usefulness of introducing SensorML in the prototype design.

CORTEX Publish-Subscribe CF and Context CF

The CORTEX Publish-Subscribe CF is not integrated with our prototype implementation due to device problems. With PDAs working in wireless ad-hoc mode, the prototype should be integrated with PocketFlow and tested in a "pervasive" environment.

Design and implementation of the CWIS component

The CWIS component is described in the architecture (11.4.7), but is not elaborated in the design and implementation of PocketFlow. A design and implementation of this component would be useful in order to gain further understanding of the architecture and a more useful prototype.

Server-based enactment service

The server-based enactment service is described as part of the architecture in Chapter 11.4.12. This part of our architecture needs to be addressed in future implementations of PocketFlow. The retrieval of workflow processes for workflow clients is currently simulated.

Support for distributed knowledge

Both the sentient object model and the intelligent artifact paradigm rely on a distributed knowledge base. Workflow clients should have part of their knowledge base shared and distributed over all entities participating in the workflow enactment.

Mobility and networking

We have not looked into mobility and networking problems when executing adaptive mobile work processes. This should be considered for future work since a real-life application is very dependent on this.

16.2 Deployment and Testing

When the implementation is updated according to the remaining work described in Chapter 16.1, testing needs to be done in a lab-setting with PDAs and laptops simulating a pervasive environment. For this testing, IPAQ PDAs would be a natural choice since they were used during the development of the CORTEX middleware. We encountered problems with our Fujitsu Siemens PDAs (13.4) when using them with this middleware.

Future testing of PocketFlow will explore and measure in more detail how successfully a workflow client is at integrating contextual information into the enactment, and how well it handles an increase in context-source subscriptions. Answers to these and related questions are important to understand the usefulness of this research.

To further improve the stability and usefulness of the PocketFlow prototype, more unit tests needs to be created. While 100% code coverage is unrealistic, developers can with good code coverage, refactor the prototype with confidence since the unit tests will catch any problems introduced.

16.3 Exploration of Social Aspects

Social aspects often receive little attention when dealing with context in context-aware systems. This may lead to problems with users that are not willing to adopt these systems. Bellotti [6] argues that the human aspects of context means that context-aware systems cannot be made to act on our behalf and that users must interact with these systems.

Real-life situations often involve a high degree of variability, where different outcomes depend on different conditions. Since it is hard to model outcomes in advance, allowing the system to take actions based on context may propose great risks when human participants are involved. Due to this, Bellotti argues that systems cannot execute autonomically based only on context-awareness, but must involve users in action outcomes. Bellotti proposes two key features to be supported in any context-aware infrastructure: intelligibility and accountability [6]:

Intelligibility Context-aware systems that seek to act upon what they infer about the context must be able to represent to their users what they know, how they know it, and what they are doing about it.

Accountability Context-aware systems must enforce user accountability when, based upon their inferences about the social context, they seek to mediate user actions that impact others.

Another social aspect of context in context-aware systems involves privacy, which may be an major problem in future applications. In a working environment though, we want to make much information available to ease the coordination of activities and add safety to the working environment. This is not personal information, and issues related to privacy are less of a concern in this setting.

Satyanarayanan [48] describes privacy as the Achilles heel of pervasive computing. Realisation of the visions of Weiser [60] suffers from problems with privacy as acknowledged

by the author. For a system to infer actions on the user's behalf, as much information as possible must be available to the system to minimise the need for explicit interaction. This need will in many cases conflict with privacy issues.

These social aspects need further exploration in order to successfully deploy context-aware systems in a pervasive environment.

Chapter 17

Conclusion

This chapter provides conclusions and a summary of our findings during the work on this report.

17.1 Workflow for Mobile Workers

A complete implementation of the WfMC reference model is probably too advanced to be of any use for mobile workers. It is in our opinion better to support a basic subset of the reference model capable of enacting one workflow process at a time. By additionally making the enactment context-aware it should be possible to build simple yet powerful applications.

17.2 Context-Awareness in Workflow

We believe that using context in workflow process enactment is feasible and will be increasingly important in the future due to improvements in sensor technology in recent years. Development of prototypes supporting context-aware workflow processes are therefore important research which may speed up development of this new technology.

This technology is not mature, despite improvements in sensor technology. Ad-hoc WLAN is unstable, something we experienced during implementation. Hard work is needed in order to make this technology reliable and user friendly.

We also found, during the design of PocketFlow, that context can be integrated into a workflow system without having to extend the WfMC reference model. Nødtvedt and Nguyen had another approach in their master thesis and concluded that an extension is needed (context information integration interface) [39]. By using standard constructs such as workflow applications, exception handlers, and workflow clients in concert we achieve the same goal. We believe this is a better approach since any workflow enactment system potentially can be made context-aware

17.3 Adaptive Workflow

An adaptive workflow system must be able to respond to context changes by generating activities which update the workflow enactment according to specified rules and facts in

an expert system. These workflow changes must be communicated to other workflow clients. If each workflow client have its own expert system with a knowledge base and this knowledge is distributed, such adaptive workflow may give rise to learning among the workflow clients in the environment. We believe an expert system based adaptive workflow enactment service is a good solution to the problem, and that the PocketFlow prototype should be able to solve this.

17.4 PocketFlow

We strongly believe the PocketFlow prototype is a good foundation for further exploration of context-aware adaptive workflow systems. Since it is developed for PDAs, it is easy to deploy in a realistic environment. Testing in a lab setting should give more answers to the usefulness of this research.

The PocketFlow is currently able to interpret and enact adaptive context-aware workflow processes on a PDA. This workflow process enactment is able to generate new activities based on a set of rules in an expert system and integrate them into the running workflow process. While we are not able to connect to real context-sources as mentioned in Chapter 13.4, the prototype is able to interpret context information in a meaningful way.

Part VIII

Appendix

Appendix A

Tools

Here we present the tools used when developing the PocketFlow proof-of-concept prototype.

A.1 Programming Tools

Programming tools include all tools used during development.

A.1.1 Embedded Visual Studio

Embedded Visual Studio is a specialised version of Visual Studio based on Visual Studio 6. It only supports development in Embedded Visual C++. Embedded Visual Studio 3 supports PocketPC 2002 development, while Embedded Visual Studio 4 supports PocketPC 2003 and PocketPC 2003 SE development. Since we must code in eVC++, Visual Studio 2005 would have been the perfect choice, but sadly it is still in production and the current beta version (as of 2005.02.8) is not stable enough for development. Hence we must choose Embedded Visual Studio 4.

A.1.2 CCCC

The C and C++ Code Counter (CCCC)[1] is an effective tool for analysing code. It gives an overview of useful software metrics related to object oriented design and structural dependencies.

A.1.3 Doxygen

Doxygen is a documentation system for C++, C, Java, and other programming languages. It can generate documentation from documented source files and extract code structure from undocumented source files.

A.2 Modelling Tools

Modelling tools are used for UML modelling, flowcharts and ad-hoc diagrams.

[1] http://sourceforge.net/projects/cccc

A.2.1 Together Architect

Borland's Together Architect is a tool for designing software solutions using UML. It supports round-trip engineering which is very useful when using it with Visual Studio.

A.2.2 Visio

Microsoft Visio is used for high-level design when UML is not suitable. The UML implementation in Visio 2003 is not good enough, but Visio provide support for other types of diagrams.

Appendix B

Class Diagrams

This appendix present the main class diagrams for the PocketFlow implementation. For more details, see the source code and documentation on the accompanying CD-ROM (appendix I).

B.1 WCIS Package

Figure B.1 shows the most important classes in the Workflow Context Integration Service and the relationships among them.

B.2 FWES and WCIS Connection

Figure B.2 shows the connection between the FWES and the WCIS. Note that the connection is implemented purely through interfaces.

B.3 FWES Package

Figure B.3 gives an overview of the main Fragment Workflow Enactment Service classes and their relationships.

B.3.1 Enactment Representation Package

Figure B.4 shows the class diagram for the enactment representation.

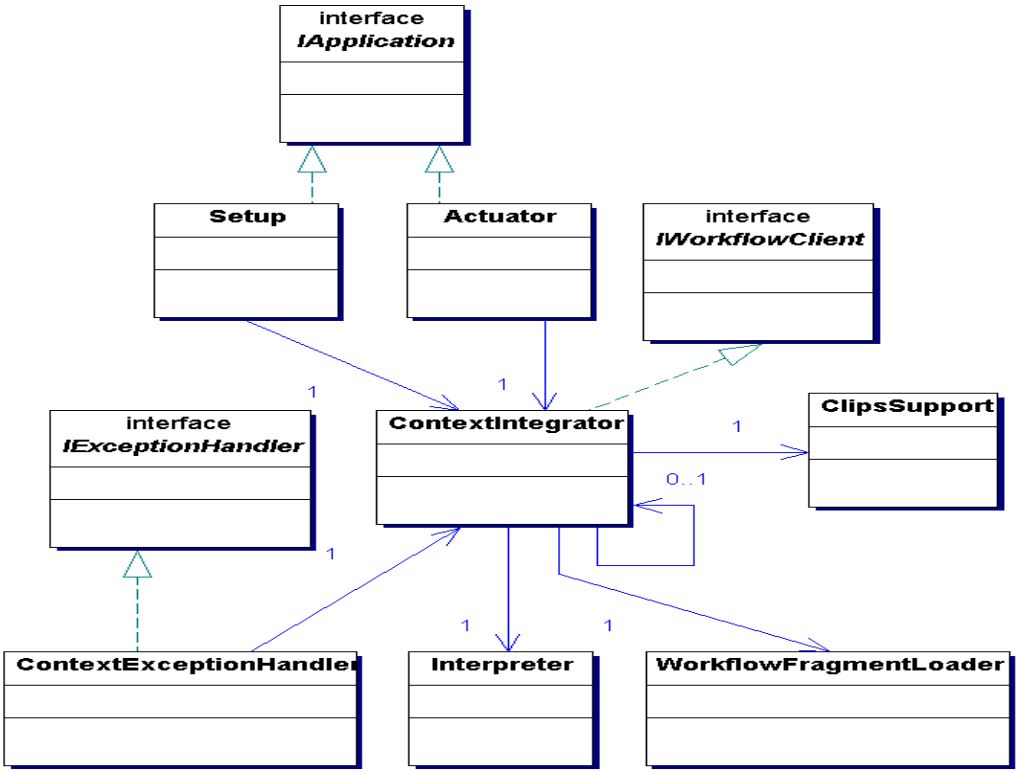

Figure B.1: WCIS class diagram

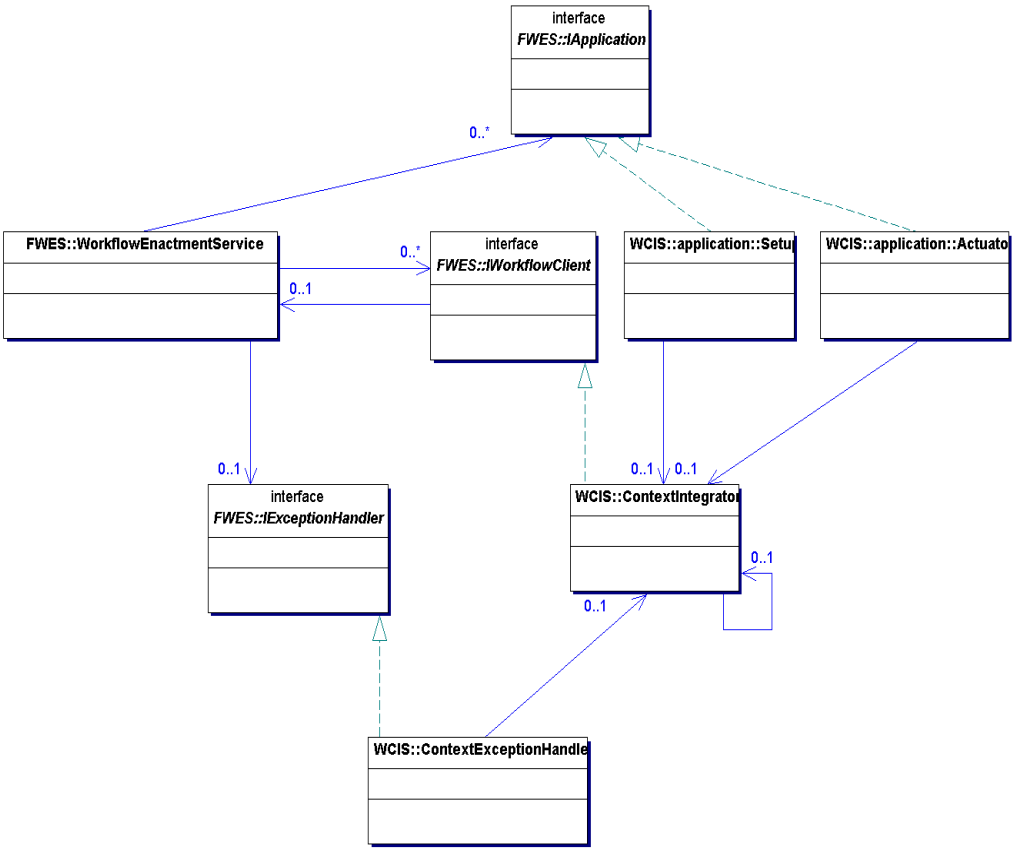

Figure B.2: FWES and WCIS connection class diagram

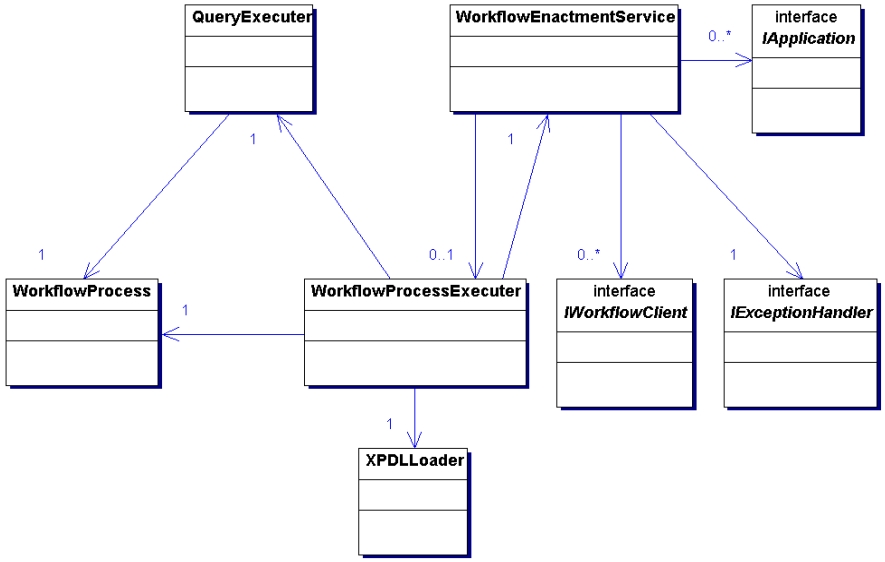

Figure B.3: FWES class diagram

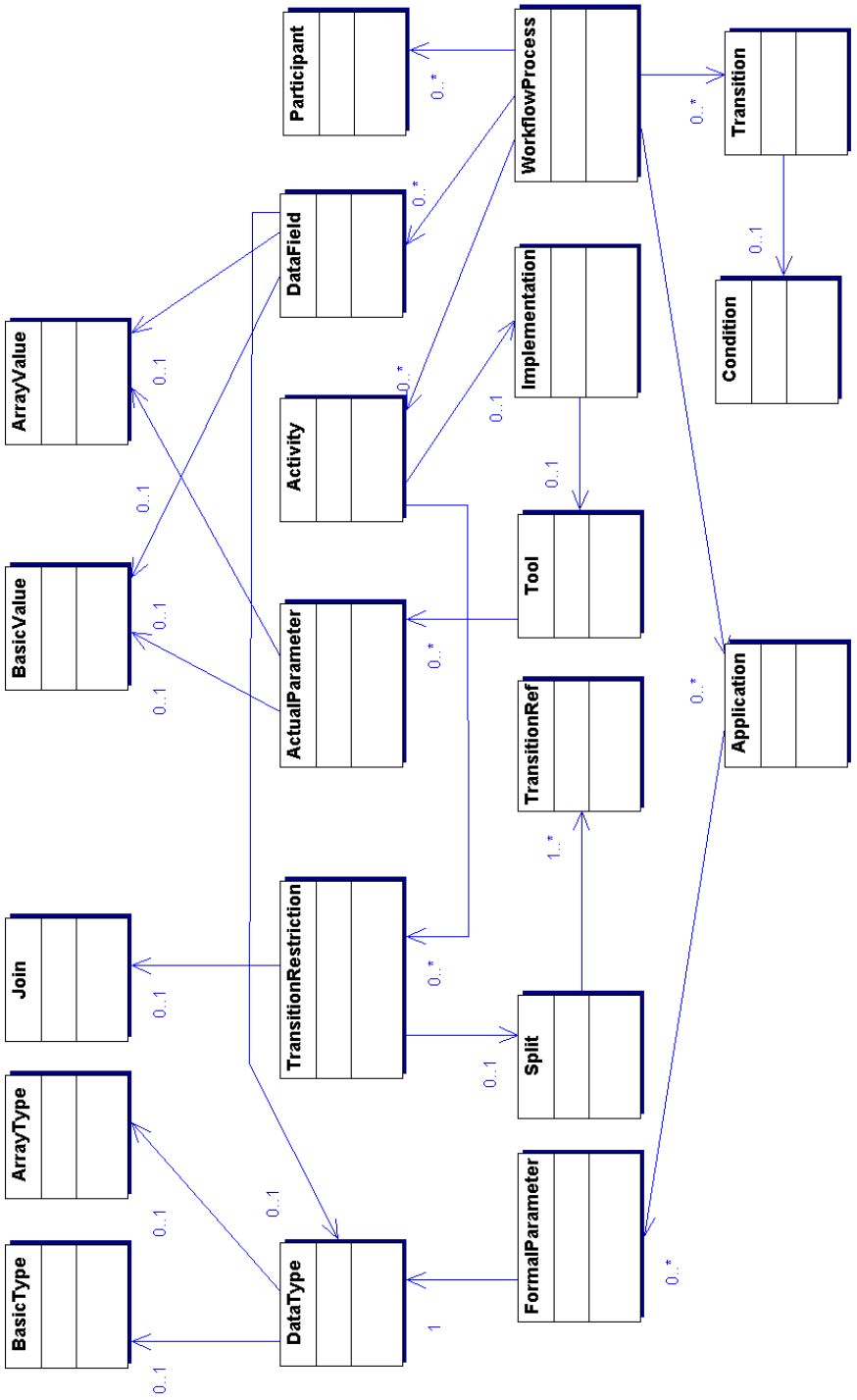

Figure B.4: Enactment representation class diagram

Appendix C

Workflow Fragments XML Schema

Listing C shows the XML schema used for the workflow fragments XML file. The any element at line 12 is any valid XPDL element.

Listing C.1: Workflow fragments XML schema

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <!--
3      Schema for Workflow Fragments xml file.
4  -->
5  <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
6      <xsd:element name="WorkflowFragment">
7          <xsd:attribute name="id" type="xsd:string" use="required" />
8          <xsd:attribute name="type" type="xsd:string" use="required" />
9          <xsd:complexType>
10             <xsd:sequence>
11                 <xsd:element ref="Depends" minOccurs="0" maxOccurs="1" />
12                 <xsd:any minOccurs="1" maxOccurs="1" />
13             </xsd:sequence>
14         </xsd:complexType>
15     </xsd:element>
16     <xsd:element name="Depends">
17         <xsd:complexType>
18             <xsdd:element ref="WorkflowFragmentRef" minOccurs="0" />
19         </xsd:complexType>
20     </xsd:element>
21     <xsd:element name="WorkflowFragmentRef">
22         <xsd:attribute name="id" type="xsd:string" use="required" />
23     </xsd:element>
24 </xsd:schema>
```

Appendix D

Actual Parameters XML Schema

Listing D shows the XML schema we used to replace the original XPDL schema for the ActualParameters element. This was done because XPDL does not support sending arrays as actual parameters.

Listing D.1: Actual parameters XML schema

```
<?xml version="1.0" encoding="UTF-8"?>
<!--
    Schema for the ActualParameters XPDL element.
    -->
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <xsd:element name="ActualParameters">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element ref="ActualParameter" minOccurs="0" />
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
    <xsd:element name="ActualParameter">
        <xsd:attribute name="Index" type="xsd:integer" />
        <xsd:complexType>
            <xsd:choice>
                <xsd:element ref="ArrayValue" minOccurs="0" />
                <xsd:element ref="BasicValue" minOccurs="0" />
            </xsd:choice>
        </xsd:complexType>
    </xsd:element>
    <xsd:element name="ArrayValue">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element ref="ArrayItem" minOccurs="0" />
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
    <xsd:element name="BasicValue" type="xsd:string" />
    <xsd:element name="ArrayItem" type="xsd:string" />
</xsd:schema>
```

Appendix E

XPDL Template Example

Listing F.1 shows an example workflow process. We call this workflow process context-aware since it uses a WCIS workflow application.

Listing E.1: XPDL example XML document

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <Package xmlns="http://www.wfmc.org/2002/XPDL1.0" xmlns:xpdl="http://www.wfmc.org
       /2002/XPDL1.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xyz="
       http://www.xyzeorder.com/workflow" xsi:schemaLocation="http://www.wfmc.org/2002/
       XPDL1.0 http://wfmc.org/standards/docs/TC-1025_schema_10_xpdl.xsd" Id="0" Name="
       sample workflow process">
3      <PackageHeader>
4          <XPDLVersion>0.09</XPDLVersion>
5          <Vendor>mowass</Vendor>
6      </PackageHeader>
7      <WorkflowProcesses>
8          <WorkflowProcess Id="1" Name="paintjob" AccessLevel="PUBLIC">
9              <ProcessHeader />
10             <FormalParameters />
11             <DataFields>
12                 <DataField Id="roomTemperatureReading" IsArray="false">
13                     <DataType>
14                         <BasicType Type="FLOAT" />
15                     </DataType>
16                     <InitialValue>-1</InitialValue>
17                 </DataField>
18             </DataFields>
19             <Participants>
20                 <Participant Id="painter">
21                     <ParticipantType Type="HUMAN" />
22                     <Description>Person in room</Description>
23                 </Participant>
24             </Participants>
25             <Applications>
26                 <!-- WCIS = Workflow Context Integration Service -->
27                 <Application Id="WCISSetup">
28                     <FormalParameters>
```

```
29                      <FormalParameter Id="sensorLookupTable" Index="1" Mode="IN">
30                          <DataType>
31                              <ArrayType DataTypes="STRING" />
32                          </DataType>
33                      </FormalParameter>
34                      <FormalParameter Id="actuatorLookupTable" Index="2" Mode="IN">
35                          <DataType>
36                              <ArrayType DataTypes="STRING" />
37                          </DataType>
38                      </FormalParameter>
39                  </FormalParameters>
40              </Application>
41          </Applications>
42          <Activities>
43              <Activity Id="init">
44                  <Description>initialize applications</Description>
45                  <Implementation>
46                      <Tool Id="WCISSetup" Type="APPLICATION">
47                          <ActualParameters>
48                              <!-- this is an extension to the XPDL standard! -->
49                              <!-- http://www.openbusinessengine.org/wiki/Wiki.jsp?page=
                                    XPDLIssues: 55.) The DataField IsArray attribute provides
                                     for array-typed process attributes, but (neglecting the
                                    deprecated ArrayType) it is impossible to describe a
                                    FormalParameter as an array type, or even to define a
                                    DeclaredType which is an array. These limitations make it
                                    impossible to pass arrays to tools or sub-flows. MUST --
                                    >
50                              <ActualParameter Index="1">
51                                  <ArrayValue>
52                                      <ArrayItem>roomTemperatureReading</ArrayItem>
53                                  </ArrayValue>
54                              </ActualParameter>
55                          </ActualParameters>
56                      </Tool>
57                  </Implementation>
58              </Activity>
59              <Activity Id="paint">
60                  <Description>manual paint job</Description>
61                  <Implementation>
62                      <No/>
63                  </Implementation>
64                  <Performer>painter</Performer>
65              </Activity>
66          </Activities>
67          <Transitions>
68              <Transition Id="1" From="init" To="paint">
69                  <Condition TYPE="CONDITION">getDataFieldValue("
                        roomTemperatureReading") > 20</Condition>>
70              </Transition>
71          </Transitions>
72      </WorkflowProcess>
```

```
73      </WorkflowProcesses>
74  </Package>
```

Appendix F

Workflow Fragments File Example

Listing F.1 shows an example workflow fragments file.

Listing F.1: Workflow fragments file example

```
<?xml version="1.0" encoding="UTF-8"?>
<!--
The fragments have access to this external information:
Transitions:
    First transition use the From value from the Transition exception.
    Last transition use the To value from the Transition exception.

Activities:
    May use the performer from the Activity found by using the From value from the Transition
        exception.

Transitions:
    May use the condition from the Transition exception.
-->
<WorkflowFragements>
    <WorkflowFragment id="splitter1" type="ACTIVITY">
        <Activity Id="splitter1">
            <Description>just a simple splitter</Description>
            <Route />
            <TransitionRestrictions>
                <TransitionRestriction>
                    <Split Type="AND">
                        <TransitionRefs>
                            <TransitionRef Id="transition2"/>
                            <TransitionRef Id="transition3"/>
                        </TransitionRefs>
                    </Split>
                </TransitionRestriction>
            </TransitionRestrictions>
        </Activity>
    </WorkflowFragment>

    <WorkflowFragment id="turnOnHeater1" type="ACTIVITY">
```

```
33          <Activity Id="turnOnHeater1">
34              <Description>manual activity for turning on heater 1</Description>
35              <Implementation>
36                  <No/>
37              </Implementation>
38              <Performer>?</Performer>
39          </Activity>
40      </WorkflowFragment>
41
42      <WorkflowFragment id="turnOnHeater2" type="ACTIVITY">
43          <Depends>
44              <WorkflowFragmentRef id="WCISActuator" />
45          </Depends>
46          <Activity Id="turnOnHeater2">
47              <Implementation>
48                  <Tool Id="WCISActuator" Type="APPLICATION">
49                      <!-- this is an extension to the XPDL standard! -->
50                      <ActualParameters>
51                          <ActualParameter Index="1">
52                              <BasicValue>heater2PowerSwitch</BasicValue>
53                          </ActualParameter>
54                          <ActualParameter Index="2">
55                              <BasicValue>on</BasicValue>
56                          </ActualParameter>
57                      </ActualParameters>
58                  </Tool>
59              </Implementation>
60          </Activity>
61      </WorkflowFragment>
62
63      <WorkflowFragment id="turnOffHeater1" type="ACTIVITY">
64          <Activity Id="turnOffHeater1">
65              <Implementation>
66                  <No/>
67              </Implementation>
68              <Performer>?</Performer>
69          </Activity>
70      </WorkflowFragment>
71
72      <WorkflowFragment id="turnOffHeater2" type="ACTIVITY">
73          <Depends>
74              <WorkflowFragmentRef id="WCISActuator" />
75          </Depends>
76          <Activity Id="turnOffHeater2">
77              <Implementation>
78                  <Tool Id="WCISActuator" Type="APPLICATION">
79                      <!-- this is an extension to the XPDL standard! -->
80                      <ActualParameters>
81                          <ActualParameter Index="1">
82                              <BasicValue>heater2PowerSwitch</BasicValue>
83                          </ActualParameter>
84                          <ActualParameter Index="2">
```

```
                        <BasicValue>off</BasicValue>
                    </ActualParameter>
                </ActualParameters>
            </Tool>
        </Implementation>
    </Activity>
</WorkflowFragment>

<WorkflowFragment id="merge1" type="ACTIVITY">
    <Activity Id="merge1">
        <Route/>
        <TransitionRestrictions>
            <TransitionRestriction>
                <Join Type="AND" />
            </TransitionRestriction>
        </TransitionRestrictions>
    </Activity>
</WorkflowFragment>

<WorkflowFragment id="transition1" type="TRANSITION">
    <Transition Id="transition1" From="?" To="splitter1" />
</WorkflowFragment>

<WorkflowFragment id="transition2" type="TRANSITION">
    <Transition Id="transition2" From="splitter1" To="turnOnHeater1" />
</WorkflowFragment>

<WorkflowFragment id="transition3" type="TRANSITION">
    <Transition Id="transition3" From="splitter1" To="turnOnHeater2" />
</WorkflowFragment>

<WorkflowFragment id="transition4" type="TRANSITION">
    <Transition Id="transition4" From="turnOnHeater1" To="turnOffHeater1">
        <Condition TYPE="CONDITION">?</Condition>>
    </Transition>
</WorkflowFragment>

<WorkflowFragment id="transition5" type="TRANSITION">
    <Transition Id="transition5" From="turnOnHeater2" To="turnOffHeater2">
        <Condition TYPE="CONDITION">?</Condition>
    </Transition>
</WorkflowFragment>

<WorkflowFragment id="transition6" type="TRANSITION">
    <Transition Id="transition6" From="turnOffHeater1" To="merge1" />
</WorkflowFragment>

<WorkflowFragment id="transition7" type="TRANSITION">
    <Transition Id="transition7" From="turnOffHeater2" To="merge1" />
</WorkflowFragment>

<WorkflowFragment id="transition8" type="TRANSITION">
```

```
137         <Transition Id="transition7" From="merge1" To="?" />
138     </WorkflowFragment>
139
140     <WorkflowFragment id="heater1IsOn" type="DATAFIELD">
141         <DataField Id="heater1IsOn" IsArray="false">
142             <DataType>
143                 <BasicType Type="BOOLEAN" />
144             </DataType>
145             <InitialValue>false</InitialValue>
146         </DataField>
147     </WorkflowFragment>
148
149     <WorkflowFragment id="heater2IsOn" type="DATAFIELD">
150         <DataField Id="heater2IsOn" IsArray="false">
151             <DataType>
152                 <BasicType Type="BOOLEAN" />
153             </DataType>
154             <InitialValue>false</InitialValue>
155         </DataField>
156     </WorkflowFragment>
157
158     <WorkflowFragment id="WCISActuator" type="APPLICATION">
159         <Application Id="WCISActuator">
160             <FormalParameters>
161                 <FormalParameter Id="actuatorId" Index="1" Mode="IN">
162                     <DataType>
163                         <BasicType Type="STRING" />
164                     </DataType>
165                 </FormalParameter>
166                 <FormalParameter Id="actuatorCommand" Index="2" Mode="IN">
167                     <DataType>
168                         <BasicType Type="STRING" />
169                     </DataType>
170                 </FormalParameter>
171             </FormalParameters>
172         </Application>
173     </WorkflowFragment>
174 </WorkflowFragements>
```

Appendix G

CLIPS Knowledge Base

Listing G shows the knowledge base used during workflow enactment in PocketFlow. It is very simple and used only to test the generation of activities in the workflow enactment.

Listing G.1: CLIPS knowledge base

```
;;*************************************************************************
;; Simple Knowledge base for a work process which controls temperature
;;*************************************************************************


(deftemplate ciTemplate
    (slot contextIntegrator (type EXTERNAL-ADDRESS))
)

; Definition of facts
(deftemplate room_facts
    (slot heater (type SYMBOL) (default off))     ; Actuator: heater status
    (slot temperature (type INTEGER) (default 0))    ; Room temperature sensor reading
    (slot maxTemperature (type INTEGER) (default 20))   ; Ideal room temperature
    (slot paintingRobot (type SYMBOL) (default ready))   ; The state of the painting robot
    (slot workflowFragmentId (type INTEGER) (default 0))   ; Workflow fragment
    (slot generateActivity (type SYMBOL) (default yes))   ; Activity generation only once
)

; Initial values for room facts
(deffacts room_facts
    (heater off)
    (temperature 0)
    (maxTemperature 20)
    (paintingRobot ready)
)

; Initial value for generateActivity
(deffacts generateActivity_fact
    (generateActivity yes)
)

```

```
; Rule for turning heater on when temperature is low (Actuation)
(defrule rule-turn-heater-on
    ?f1 <- (heater off)
    (room_facts (maxTemperature ?maxTemp))
    (room_facts (temperature ?temp))
    (> ?maxTemp ?temp)
    ?f2 <- (generateActivity yes)
    ?exception <- (transitionException ?transitionId)
    =>
    (assert (heater on))
    (retract ?f1)
    (retract ?f2)
    (printout t "Heater is turned on")
    (generateNewActivity "transition1" ?transitionId)
    (retract ?exception)
)

; Rule for turning heater off when temperature is high (Actuation)
(defrule rule-turn-heater-Off
    ?f1 <- (heater on)
    (room_facts (maxTemperature ?maxTemp))
    (room_facts (temperature ?temp))
    (> ?maxTemp ?temp)
    =>
    (assert (heater off))
    (retract ?f1)
    (printout t "heater is turned off")
    (generateNewActivity workflowFragmentId)
)

; Rule for testing
(defrule rule-testing
    (wearetesting true)
    (ciTemplate (contextIntegrator ?value))
    =>
    (printout t "testing started...")
    (printout t ?value)
)
```

Appendix H

Context-Aware Workflow Example

The log file in Listing H.1 is printed to the file system when enacting the context-aware workflow shown in Figure H.1. This is a simple workflow where the first activity sets up sensors and actuators by using a context-aware workflow application. After this is done the transition to the paint activity is evaluated. Since the "roomTemperatureReading" data field currently has the value "-1" a transition exception is created. This transition exception is caught by the CLIPS expert system which generates new activities. The workflow will now look like Figure H.2 and is enacted. We have used the GUI to control the manual part of the workflow process.

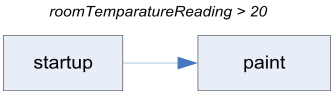

Figure H.1: Context-aware workflow

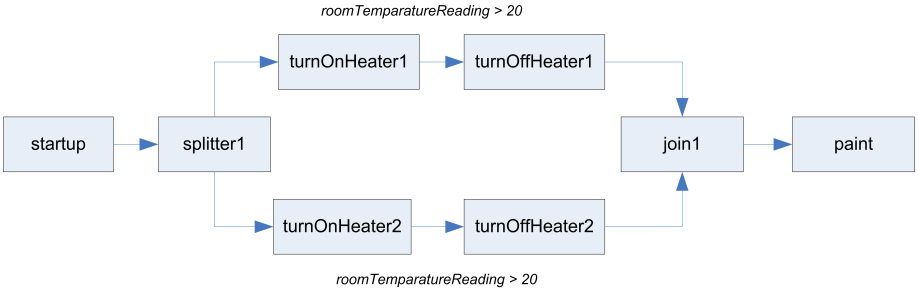

Figure H.2: Context-aware workflow after transition exception

See Appendix E for the XPDL template, Appendix F for the workflow fragments, and Appendix G for the CLIPS knowledge base used in this workflow.

Listing H.1: Log file

```
1 ================================================
2 Log is started on 06.06.2005, at 16:18:11:000, executable: \GUI.exe (ProcID: 0x52db727e),
      compile time : Jun 6 2005 14:53:00
```

```
3  ContextIntegrator::start()
4  WorkflowFragmentLoader::setFragmentsFilePath()
5  WorkflowFragmentLoader::setFragmentsFilePath(); File path = \workflowFragments.xml
6  ContextIntegrator::initialiseExpertSystem()
7  ClipsSupport::initializeEnvironment()
8  ClipsSupport::defineFunction2(); Function name: generateNewActivity
9  ClipsSupport::load(); File name: \KnowledgeBase.CLP
10 ClipsSupport::reset()
11 ContextIntegrator::ContextIntegratorThread(); Thread started
12 WorkflowEnactmentService::loadWorkflowProcess()
13 WorkflowProcessExecuter::loadProcess()
14 WorkflowProcessExecuter::start()
15 WorkflowEnactmentService::notifyWorkflowStatusChange()
16 WorkflowProcessExecuter::ProcessExecuterThread(); Thread was started
17 WorkflowProcessExecuter::addRunningActivity(); Activity Id: init
18 WorkflowEnactmentService::registerActivity()
19 WorkflowEnactmentService::notifyActivityStarting()
20 WorkflowProcessExecuter::ProcessExecuterThread(); Has more activities or waiting transitions
21 WorkflowProcessExecuter::ActivityExecuterThread(); Thread started
22 WorkflowProcessExecuter::ActivityExecuterThread(); IMPLEMENTATION_TYPE_TOOL
23 WorkflowProcessExecuter::ActivityExecuterThread(); TOOL_TYPE_APPLICATION
24 WorkflowProcessExecuter::executeApplication()
25 WorkflowEnactmentService::executeApplication()
26 WorkflowEnactmentService::executeApplication(); Trying to find the application with Id "
       WCISSetup"
27 Setup::application_execute(); WorkflowProcess Id: 1
28 ContextIntegrator::subscribeToSensor()
29 WorkflowProcessExecuter::ActivityExecuterThread(); Activity Id=init
30 WorkflowProcessExecuter::ActivityExecuterThread(); Thread ended Activity Id=init
31 WorkflowProcessExecuter::ProcessExecuterThread();
       THRD_MESSAGE_NOTIFY_COMPLETED_ACTIVITY
32 WorkflowProcessExecuter::evaluateActivityTransition(); Activity Id: init
33 WorkflowProcessExecuter::evaluateActivityTransition(); No transition restrictions
34 WorkflowProcessExecuter::evaluateTransition(); Transitions id=1, from=init, to=paint
35 ScriptSupport(); Initializing Lua
36 LuaWorkflowProcess::evaluateCondition()
37 ScriptSupport::doString(); Lua script: wp_result = (getDataFieldValue("
       roomTemperatureReading") > 20)
38 LuaWorkflowProcess::lua_getDataFieldValue()
39 LuaWorkflowProcess::lua_getDataFieldValue(); DataField Id==roomTemperatureReading
40 LuaWorkflowProcess::getDataFieldValue(); DataType was BasicType
41 ScriptSupport(); Closing Lua
42 WorkflowProcessExecuter::evaluateTransition(); Transition condition not satisfied.
43 WorkflowProcessExecuter::removeRunningActivity(); Activity Id: init
44 WorkflowEnactmentService::notifyActivityEnded()
45 WorkflowProcessExecuter::ProcessExecuterThread(); Has more activities or waiting transitions
46 WorkflowProcessExecuter::ProcessExecuterThread();
       THRD_MESSAGE_NOTIFY_TRANSITION_EXCEPTION
47 ContextExceptionHandler::handleException()
48 ContextExceptionHandler::handleException(); Message: whoho! FIXME, Transition Id: 1
49 ContextIntegrator::handleTransitionException()
50 ClipsSupport::assertString(); Fact: (transitionException "1")
```

```
51 ClipsSupport::run()
52 ClipsSupport; From stdout: Heater is turned on
53 ContextIntegrator::clipsGenerateNewActivity()
54 ContextIntegrator::clipsGenerateNewActivity(); Fragment id: transition1, Original transition id:
       1
55 WorkflowProcessExecuter::ProcessExecuterThread(); Has more activities or waiting transitions
56 ContextIntegrator::ContextIntegratorThread(); Create new activity, Transition fragment id=
       transition1, Original transition id=1
57 ContextIntegrator::generateNewActivity()
58 WorkflowProcessExecuter::suspend()
59 WorkflowEnactmentService::notifyWorkflowStatusChange()
60 WorkflowFragmentLoader::loadFirstTransition()
61 WorkflowFragmentLoader::loadTransition(); Transition fragment Id: transition1
62 WorkflowFragmentLoader::loadDependencies()
63 WorkflowFragmentLoader::loadTransition(); Adding transition with Id: transition1
64 WorkflowFragmentLoader::loadActivity(); Activity fragment Id: splitter1
65 WorkflowFragmentLoader::loadDependencies()
66 WorkflowFragmentLoader::loadActivity(); Has transition restrictions
67 WorkflowFragmentLoader::loadTransition(); Transition fragment Id: transition2
68 WorkflowFragmentLoader::loadDependencies()
69 WorkflowFragmentLoader::loadTransition(); Adding transition with Id: transition2
70 WorkflowFragmentLoader::loadActivity(); Activity fragment Id: turnOnHeater1
71 WorkflowFragmentLoader::loadDependencies()
72 WorkflowFragmentLoader::loadActivity(); No split transition restriction
73 WorkflowFragmentLoader::loadTransitionByFrom(); From: turnOnHeater1
74 WorkflowFragmentLoader::loadDependencies()
75 WorkflowFragmentLoader::loadTransition(); Adding transition with Id: transition4
76 WorkflowFragmentLoader::loadActivity(); Activity fragment Id: turnOffHeater1
77 WorkflowFragmentLoader::loadDependencies()
78 WorkflowFragmentLoader::loadActivity(); No split transition restriction
79 WorkflowFragmentLoader::loadTransitionByFrom(); From: turnOffHeater1
80 WorkflowFragmentLoader::loadDependencies()
81 WorkflowFragmentLoader::loadTransition(); Adding transition with Id: transition6
82 WorkflowFragmentLoader::loadActivity(); Activity fragment Id: merge1
83 WorkflowFragmentLoader::loadDependencies()
84 WorkflowFragmentLoader::loadActivity(); Has transition restrictions
85 WorkflowFragmentLoader::loadActivity(); No split transition restriction
86 WorkflowFragmentLoader::loadTransitionByFrom(); From: merge1
87 WorkflowFragmentLoader::loadDependencies()
88 WorkflowFragmentLoader::loadTransition(); Adding transition with Id: transition7
89 WorkflowFragmentLoader::loadTransition(); Transition fragment Id: transition3
90 WorkflowFragmentLoader::loadDependencies()
91 WorkflowFragmentLoader::loadTransition(); Adding transition with Id: transition3
92 WorkflowFragmentLoader::loadActivity(); Activity fragment Id: turnOnHeater2
93 WorkflowFragmentLoader::loadDependencies()
94 WorkflowFragmentLoader::loadDependencies(): Has dependency with id: WCISActuator
95 WorkflowFragmentLoader::loadDependencies(); Found fragment with id: WCISActuator and
       type: APPLICATION
96 WorkflowFragmentLoader::loadDependencies()
97 WorkflowFragmentLoader::loadActivity(); No split transition restriction
98 WorkflowFragmentLoader::loadTransitionByFrom(); From: turnOnHeater2
99 WorkflowFragmentLoader::loadDependencies()
```

```
100 WorkflowFragmentLoader::loadTransition(); Adding transition with Id: transition5
101 WorkflowFragmentLoader::loadActivity(); Activity fragment Id: turnOffHeater2
102 WorkflowFragmentLoader::loadDependencies()
103 WorkflowFragmentLoader::loadDependencies(): Has dependency with id: WCISActuator
104 WorkflowFragmentLoader::loadDependencies(); Found fragment with id: WCISActuator and
        type: APPLICATION
105 WorkflowFragmentLoader::loadDependencies()
106 WorkflowFragmentLoader::loadActivity(); No split transition restriction
107 WorkflowFragmentLoader::loadTransitionByFrom(); From: turnOffHeater2
108 WorkflowFragmentLoader::loadDependencies()
109 WorkflowFragmentLoader::loadTransition(); Adding transition with Id: transition7
110 WorkflowFragmentLoader::loadActivity(); Activity fragment Id: merge1
111 WorkflowFragmentLoader::loadActivity(); The activity fragment was already loaded.
112 WorkflowEnactmentService::updateTransition()
113 WorkflowProcessExecuter::evaluateNewTransition()
114 WorkflowProcessExecuter::resume()
115 WorkflowEnactmentService::notifyWorkflowStatusChange()
116 WorkflowProcessExecuter::ProcessExecuterThread();
        THRD_MESSAGE_NOTIFY_EVALUATE_NEW_TRANSITION
117 WorkflowProcessExecuter::evaluateTransition(); Transitions id=transition1, from=init, to=
        splitter1
118 WorkflowProcessExecuter::evaluateTransition(); No condition, setting satisfied = true
119 WorkflowProcessExecuter::addRunningActivity(); Activity Id: splitter1
120 WorkflowEnactmentService::registerActivity()
121 WorkflowEnactmentService::notifyActivityStarting()
122 WorkflowProcessExecuter::ProcessExecuterThread(); Has more activities or waiting transitions
123 WorkflowProcessExecuter::ActivityExecuterThread(); Thread started
124 WorkflowProcessExecuter::ActivityExecuterThread(); IMPLEMENTATION_TYPE_NO
125 WorkflowProcessExecuter::ActivityExecuterThread(); Activity Id=splitter1
126 WorkflowProcessExecuter::ActivityExecuterThread(); Thread ended Activity Id=splitter1
127 WorkflowProcessExecuter::ProcessExecuterThread();
        THRD_MESSAGE_NOTIFY_COMPLETED_ACTIVITY
128 WorkflowProcessExecuter::evaluateActivityTransition(); Activity Id: splitter1
129 WorkflowProcessExecuter::evaluateActivityTransition(); Has transition restrictions
130 WorkflowProcessExecuter::evaluateSplitRestriction()
131 WorkflowProcessExecuter::evaluateTransition(); Transitions id=transition2, from=splitter1, to=
        turnOnHeater1
132 WorkflowProcessExecuter::evaluateTransition(); No condition, setting satisfied = true
133 WorkflowProcessExecuter::addRunningActivity(); Activity Id: turnOnHeater1
134 WorkflowEnactmentService::registerActivity()
135 WorkflowProcessExecuter::ActivityExecuterThread(); Thread started
136 WorkflowEnactmentService::notifyActivityStarting()
137 WorkflowProcessExecuter::evaluateTransition(); Transitions id=transition3, from=splitter1, to=
        turnOnHeater2
138 WorkflowProcessExecuter::evaluateTransition(); No condition, setting satisfied = true
139 WorkflowProcessExecuter::addRunningActivity(); Activity Id: turnOnHeater2
140 WorkflowEnactmentService::registerActivity()
141 WorkflowEnactmentService::notifyActivityStarting()
142 WorkflowProcessExecuter::evaluateActivityTransition(); No transition
143 WorkflowProcessExecuter::removeRunningActivity(); Activity Id: splitter1
144 WorkflowEnactmentService::notifyActivityEnded()
145 WorkflowProcessExecuter::ActivityExecuterThread(); Thread started
```

```
146 WorkflowProcessExecuter::ActivityExecuterThread(); IMPLEMENTATION_TYPE_TOOL
147 WorkflowProcessExecuter::ActivityExecuterThread(); TOOL_TYPE_APPLICATION
148 WorkflowProcessExecuter::executeApplication()
149 WorkflowEnactmentService::executeApplication()
150 WorkflowEnactmentService::executeApplication(); Trying to find the application with Id ”
        WCISActuator”
151 Actuator::application_execute()
152 ContextIntegrator::executeActuatorCommand()
153 WorkflowProcessExecuter::ActivityExecuterThread(); Activity Id=turnOnHeater2
154 WorkflowProcessExecuter::ActivityExecuterThread(); Thread ended Activity Id=turnOnHeater2
155 WorkflowProcessExecuter::ProcessExecuterThread(); Has more activities or waiting transitions
156 WorkflowProcessExecuter::ProcessExecuterThread();
        THRD_MESSAGE_NOTIFY_COMPLETED_ACTIVITY
157 WorkflowProcessExecuter::evaluateActivityTransition(); Activity Id: turnOnHeater2
158 WorkflowProcessExecuter::evaluateActivityTransition(); No transition restrictions
159 WorkflowProcessExecuter::evaluateTransition(); Transitions id=transition5, from=
        turnOnHeater2, to=turnOffHeater2
160 ScriptSupport(); Initializing Lua
161 LuaWorkflowProcess::evaluateCondition()
162 ScriptSupport::doString(); Lua script: wp_result = (getDataFieldValue(”
        roomTemperatureReading”) > 20)
163 LuaWorkflowProcess::lua_getDataFieldValue()
164 LuaWorkflowProcess::lua_getDataFieldValue(); DataField Id==roomTemperatureReading
165 LuaWorkflowProcess::getDataFieldValue(); DataType was BasicType
166 ScriptSupport(); Closing Lua
167 WorkflowProcessExecuter::evaluateTransition(); Transition condition not satisfied.
168 WorkflowProcessExecuter::removeRunningActivity(); Activity Id: turnOnHeater2
169 WorkflowEnactmentService::notifyActivityEnded()
170 WorkflowProcessExecuter::ProcessExecuterThread(); Has more activities or waiting transitions
171 WorkflowProcessExecuter::ProcessExecuterThread();
        THRD_MESSAGE_NOTIFY_TRANSITION_EXCEPTION
172 ContextExceptionHandler::handleException()
173 ContextExceptionHandler::handleException(); Message: whoho! FIXME, Transition Id:
        transition5
174 ContextIntegrator::handleTransitionException()
175 ClipsSupport::assertString(); Fact: (transitionException ”transition5”)
176 ClipsSupport::run()
177 WorkflowProcessExecuter::ProcessExecuterThread(); Has more activities or waiting transitions
178 WorkflowProcessExecuter::ProcessExecuterThread();
        THRD_MESSAGE_NOTIFY_DATA_FIELD_UPDATED
179 WorkflowProcessExecuter::evaluateDataField()
180 ScriptSupport(); Initializing Lua
181 LuaWorkflowProcess::evaluateCondition()
182 ScriptSupport::doString(); Lua script: wp_result = (getDataFieldValue(”
        roomTemperatureReading”) > 20)
183 LuaWorkflowProcess::lua_getDataFieldValue()
184 LuaWorkflowProcess::lua_getDataFieldValue(); DataField Id==roomTemperatureReading
185 LuaWorkflowProcess::getDataFieldValue(); DataType was BasicType
186 ScriptSupport(); Closing Lua
187 WorkflowProcessExecuter::evaluateDataField(); The transition is now satisfied
188 WorkflowProcessExecuter::evaluateTransition(); Transitions id=transition5, from=
        turnOnHeater2, to=turnOffHeater2
```

```
189 WorkflowProcessExecuter::addRunningActivity(); Activity Id: turnOffHeater2
190 WorkflowEnactmentService::registerActivity()
191 WorkflowEnactmentService::notifyActivityStarting()
192 WorkflowProcessExecuter::ProcessExecuterThread(); Has more activities or waiting transitions
193 WorkflowProcessExecuter::ActivityExecuterThread(); Thread started
194 WorkflowProcessExecuter::ActivityExecuterThread(); IMPLEMENTATION_TYPE_TOOL
195 WorkflowProcessExecuter::ActivityExecuterThread(); TOOL_TYPE_APPLICATION
196 WorkflowProcessExecuter::executeApplication()
197 WorkflowEnactmentService::executeApplication()
198 WorkflowEnactmentService::executeApplication(); Trying to find the application with Id "
        WCISActuator"
199 Actuator::application_execute()
200 ContextIntegrator::executeActuatorCommand()
201 WorkflowProcessExecuter::ActivityExecuterThread(); Activity Id=turnOffHeater2
202 WorkflowProcessExecuter::ActivityExecuterThread(); Thread ended Activity Id=turnOffHeater2
203 WorkflowProcessExecuter::ProcessExecuterThread();
        THRD_MESSAGE_NOTIFY_COMPLETED_ACTIVITY
204 WorkflowProcessExecuter::evaluateActivityTransition(); Activity Id: turnOffHeater2
205 WorkflowProcessExecuter::evaluateActivityTransition(); No transition restrictions
206 WorkflowProcessExecuter::evaluateTransition(); Transitions id=transition7, from=
        turnOffHeater2, to=merge1
207 WorkflowProcessExecuter::evaluateTransition(); No condition, setting satisfied = true
208 WorkflowProcessExecuter::evaluateTransition(); Has join transitions restriction
209 WorkflowProcessExecuter::evaluateJoinTransition(); Transitions id=transition7, from=
        turnOffHeater2, to=merge1
210 WorkflowProcessExecuter::evaluateTransition(); Transition with id transition7 must wait for join
211 WorkflowProcessExecuter::removeRunningActivity(); Activity Id: turnOffHeater2
212 WorkflowEnactmentService::notifyActivityEnded()
213 WorkflowProcessExecuter::ProcessExecuterThread(); Has more activities or waiting transitions
214 WorkflowEnactmentService::notifyCompletedActivity()
215 WorkflowProcessExecuter::notifyCompletedActivity();
216 WorkflowProcessExecuter::ActivityExecuterThread(); Activity Id=turnOnHeater1
217 WorkflowProcessExecuter::ActivityExecuterThread(); Thread ended Activity Id=turnOnHeater1
218 WorkflowProcessExecuter::ProcessExecuterThread();
        THRD_MESSAGE_NOTIFY_COMPLETED_ACTIVITY
219 WorkflowProcessExecuter::evaluateActivityTransition(); Activity Id: turnOnHeater1
220 WorkflowProcessExecuter::evaluateActivityTransition(); No transition restrictions
221 WorkflowProcessExecuter::evaluateTransition(); Transitions id=transition4, from=
        turnOnHeater1, to=turnOffHeater1
222 ScriptSupport(); Initializing Lua
223 LuaWorkflowProcess::evaluateCondition()
224 ScriptSupport::doString(); Lua script: wp_result = (getDataFieldValue("
        roomTemperatureReading") > 20)
225 LuaWorkflowProcess::lua_getDataFieldValue()
226 LuaWorkflowProcess::lua_getDataFieldValue(); DataField Id==roomTemperatureReading
227 LuaWorkflowProcess::getDataFieldValue(); DataType was BasicType
228 ScriptSupport(); Closing Lua
229 WorkflowProcessExecuter::evaluateTransition(); Transition condition satisfied.
230 WorkflowProcessExecuter::addRunningActivity(); Activity Id: turnOffHeater1
231 WorkflowEnactmentService::registerActivity()
232 WorkflowProcessExecuter::ActivityExecuterThread(); Thread started
233 WorkflowEnactmentService::notifyActivityStarting()
```

```
234 WorkflowProcessExecuter::removeRunningActivity(); Activity Id: turnOnHeater1
235 WorkflowEnactmentService::notifyActivityEnded()
236 WorkflowProcessExecuter::ProcessExecuterThread(); Has more activities or waiting transitions
237 WorkflowEnactmentService::notifyCompletedActivity()
238 WorkflowProcessExecuter::notifyCompletedActivity();
239 WorkflowProcessExecuter::ActivityExecuterThread(); Activity Id=turnOffHeater1
240 WorkflowProcessExecuter::ActivityExecuterThread(); Thread ended Activity Id=turnOffHeater1
241 WorkflowProcessExecuter::ProcessExecuterThread();
        THRD_MESSAGE_NOTIFY_COMPLETED_ACTIVITY
242 WorkflowProcessExecuter::evaluateActivityTransition(); Activity Id: turnOffHeater1
243 WorkflowProcessExecuter::evaluateActivityTransition(); No transition restrictions
244 WorkflowProcessExecuter::evaluateTransition(); Transitions id=transition6, from=
        turnOffHeater1, to=merge1
245 WorkflowProcessExecuter::evaluateTransition(); No condition, setting satisfied = true
246 WorkflowProcessExecuter::evaluateTransition(); Has join transitions restriction
247 WorkflowProcessExecuter::evaluateJoinTransition(); Transitions id=transition6, from=
        turnOffHeater1, to=merge1
248 WorkflowProcessExecuter::addRunningActivity(); Activity Id: merge1
249 WorkflowEnactmentService::registerActivity()
250 WorkflowEnactmentService::notifyActivityStarting()
251 WorkflowProcessExecuter::removeRunningActivity(); Activity Id: turnOffHeater1
252 WorkflowEnactmentService::notifyActivityEnded()
253 WorkflowProcessExecuter::ActivityExecuterThread(); Thread started
254 WorkflowProcessExecuter::ActivityExecuterThread(); IMPLEMENTATION_TYPE_NO
255 WorkflowProcessExecuter::ActivityExecuterThread(); Activity Id=merge1
256 WorkflowProcessExecuter::ActivityExecuterThread(); Thread ended Activity Id=merge1
257 WorkflowProcessExecuter::ProcessExecuterThread(); Has more activities or waiting transitions
258 WorkflowProcessExecuter::ProcessExecuterThread();
        THRD_MESSAGE_NOTIFY_COMPLETED_ACTIVITY
259 WorkflowProcessExecuter::evaluateActivityTransition(); Activity Id: merge1
260 WorkflowProcessExecuter::evaluateActivityTransition(); Has transition restrictions
261 WorkflowProcessExecuter::evaluateTransition(); Transitions id=transition7, from=merge1, to=
        paint
262 WorkflowProcessExecuter::evaluateTransition(); No condition, setting satisfied = true
263 WorkflowProcessExecuter::addRunningActivity(); Activity Id: paint
264 WorkflowEnactmentService::registerActivity()
265 WorkflowProcessExecuter::ActivityExecuterThread(); Thread started
266 WorkflowEnactmentService::notifyActivityStarting()
267 WorkflowProcessExecuter::removeRunningActivity(); Activity Id: merge1
268 WorkflowEnactmentService::notifyActivityEnded()
269 WorkflowProcessExecuter::ProcessExecuterThread(); Has more activities or waiting transitions
270 WorkflowEnactmentService::notifyCompletedActivity()
271 WorkflowProcessExecuter::notifyCompletedActivity();
272 WorkflowProcessExecuter::ActivityExecuterThread(); Activity Id=paint
273 WorkflowProcessExecuter::ActivityExecuterThread(); Thread ended Activity Id=paint
274 WorkflowProcessExecuter::ProcessExecuterThread();
        THRD_MESSAGE_NOTIFY_COMPLETED_ACTIVITY
275 WorkflowProcessExecuter::evaluateActivityTransition(); Activity Id: paint
276 WorkflowProcessExecuter::evaluateActivityTransition(); No transition restrictions
277 WorkflowProcessExecuter::evaluateActivityTransition(); No transition
278 WorkflowProcessExecuter::removeRunningActivity(); Activity Id: paint
279 WorkflowEnactmentService::notifyActivityEnded()
```

```
280 WorkflowEnactmentService::notifyWorkflowStatusChange()
281 WorkflowProcessExecuter::ProcessExecuterThread(); Thread ended
282 WorkflowEnactmentService::clearWorkflowProcess()
283 WorkflowEnactmentService::clearWorkflowProcess()
284 WorkflowProcessExecuter::stop()
```

Appendix I

CD-ROM

The accompanying CD-ROM constains

- the code for PocketFlow and other dependencies: Note that you must select PocketPC2003 as the target environment to be able to build the application.
- the WorkflowClient API documentation: Doxygen generated documentation for the source code for the WorkflowClient part of PocketFlow.
- the WorkflowClient CCCC documentation: The CCCC documentation described in Chapter 13.

To start the main menu select the *index.html* file and open it in your favourite web browser.

Appendix J

Glossary

Actuator

An object that can perform some action. It may be a physical object, an application invocation, or something more abstract like a workflow activity.

Activity

A description of a piece of work that forms one logical step within a process. An activity may be a manual activity, which does not support computer automation, or a workflow (automated) activity. A workflow activity requires human and/or machine resources(s) to support process execution; where human resource is required an activity is allocated to a workflow participant.

Activity Instance

The representation of an activity within a (single) enactment of a process, i.e. within a process instance.

Activity Theory

A formal theory of human work activities and a philosophical framework for studying human work practices.

Business Process

A set of one or more linked procedures or activities which collectively realise a business objective or policy goal, normally within the context of an organisational structure, defining functional roles and relationships.

CLIPS

C Language Integrated Production System.

CORTEX

CO-operating Real-time senTient objects: architecture and EXperimental evaluation.

COTS

Commercial Off The shelf Software.

CWIS

Cooperative Workflow Integration Service.

eVC++

embedded Visual C++.

FEL

Filter Event Language

FWES

Fragment Workflow Enactment Service.

MOWAHS

MObile Work Across Heterogeneous Systems.

PDA

Personal Digital Assistant.

Process Definition

The representation of a business process in a form which supports automated manipulation, such as modelling, or enactment by a workflow management system. The process definition consists of a network of activities and their relationships, criteria to indicate the start and termination of the process, and information about the individual activities, such as participants, associated IT applications and data, etc.

Process Instance

The representation of a single enactment of a process, or activity within a process, including its associated data. Each instance represents a separate thread of execution of the process or activity, which may be controlled independently and will have its own internal state and externally visible identity, which may be used as a handle, for example, to record or retrieve audit data relating to the individual enactment.

PSM

Publish-subscribe middleware.

Sentient Object

Mobile, intelligent software component which is able to sense its environment via sensors and react to sensed information via actuators.

Ubiquitous Computing

An environment where computers are integrated seamlessly into the environment, are aware of their surroundings, and can adapt their behaviour accordingly as proposed by Weiser [60].

WCIS

Workflow Client Integration Service

Workflow

The automation of a business process, in whole or part, during which documents, information or tasks are passed from one participant to another for action, according to a set of procedural rules.

Workflow Management System

A system that defines, creates, and manages the execution of workflows through the use of software, running on one or more workflow engines, which is able to interpret the process definition, interact with workflow participants and, where required, invoke the use of IT tools and applications.

WLAN

Wireless Local Area Network.

XML

eXtensible Markup Language.

XPDL

XML Process Definition Language, standard used to describe workflow.

Bibliography

[1] ACM. Association for Computing Machinery portal. http://www.portal.acm.org/, Accessed June 7th 2005.

[2] Michael Adams, David Edmond, and Arthur H.M. ter Hofstede. The application of activity theory to dynamic workflow adaptation issues. http://sky.fit.qut.edu.au/ adamsmj/The Application of Activity Theory to Dynamic Workflow Adaptation Issues.pdf.

[3] Manuel Armada and Pablo Gonzales de Santos. Climbing and walking robots for the maritim industries. http://www.ensieta.fr/naval_design/CCIV_ma.pdf.

[4] Vijayalakshmi Atluri and Soon Ae Chun. Handling Dynamic Changes in Decentralized Workflow Execution Environments. pages 813–825, 2003. LNCS 2736.

[5] Jakob E. Bardram. Plans as Situated Action: An Activity Theory Approach to Workflow Systems. In *5th European Conference on Computer Supported Cooperative Work*, Lancaster University, UK, 7-11 September 1997. Kluwer Academic Publishers.

[6] Victoria Bellotti and Keith Edwards. Intelligibility and Accountability: Human Considerations in Context-Aware Systems. *Human-Computer Interaction (HCI) Journal. Special Issue: Context-Aware Computing*, 16(2–4):193–212, 2001.

[7] Gregory Biegel and Vinny Cahill. A Framework for Developing Mobile, Context-aware Applications. In *Second IEEE International Conference on Pervasive Computing and Communications (PerCom'04)*, pages 361–365, Orlando, Florida, March 14-17 2004. IEEE Computer Society Press.

[8] Jorge Cardoso, Zongwei Luo, John Miller, Amit Sheth, and Krys Kochut. Survivability architecture for workflow management systems. In *Proceedings of the 39th Annual ACM Southeast Conference (ACMSE'01)*, pages 207–216, 2001.

[9] Shyam R. Chidamber and Chris F. Kemerer. Towards a metrics suite for object oriented design. volume 26, pages 197–211, New York, NY, USA, 1991. ACM Press.

[10] Michael Clarke, Gordon S. Blair, Geoff Coulson, and Nikolaos Parlavantzas. An Efficient Component Model for the Construction of Adaptive Middleware. In *Proc. of IFIP/ACM Middleware 2001*, Heidelberg, Germany, November 2001. IEEE CS Press.

[11] The Workflow Management Coalition. Workflow Management Coalition - Terminology & Glossary. Technical report, The Workflow Management Coalition (WfMC), February 1999. Document Number WFMC-TC-1011.

[12] David E. Culler and Wei Hong. Wireless sensor networks: Introduction. *Communications of the ACM*, 47(6):30–33, 2004.

[13] Anind K. Dey. *Providing Architectural Support for Building Context-Aware Applications*. PhD thesis, College of Computing, Georgia Institute of Technology, Atlanta, GA, USA, December 2000.

[14] Anind K. Dey and Gregory D. Abowd. Towards a Better Understanding of Context and Context-Awareness. In *Workshop on The What, Who, Where, When, and How of Context-Awareness, as part of the 2000 Conference on Human Factors in Computing Systems (CHI 2000)*, The Hague, The Netherlands, April 2000. ACM Press. Also GVU Technical Report GIT-GVU-99-22.

[15] Adrian Fitzpatrick, Gregory Biegel, Siobhán Clarke, and Vinny Cahill. Towards a Sentient Object Model. In *Workshop on Engineering Context-Aware Object Oriented Systems and Environments (ECOOSE)*, Seattle, WA, USA, November 2002.

[16] Martin Fowler. Inversion of control containers and the dependency injection pattern. http://www.martinfowler.com/articles/injection.html.

[17] Martin Fowler. *UML Distilled: A Brief Guide to the Standard Object Modeling Language*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2003.

[18] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995.

[19] Mike Grant. Introduction to extreme programming. http://www.acm.jhu.edu/peerteach/xp/xp.pdf.

[20] Saul Greenberg. Context as a Dynamic Construct. *Human-Computer Interaction (HCI) Journal. Special Issue: Context-Aware Computing*, 16(2–4):257–268, 2001.

[21] Frode Hauso and Øivind Røed. Adaptive mobile work processes, student depth project. Technical report, 2004.

[22] Jason Hill, Mike Horton, Ralph Kling, and Lakshman Krishnamurthy. The Platforms Enabling Wireless Sensor Networks. *Communications of the ACM*, 47(6):41–46, 2004.

[23] D. Hollingsworth. The Workflow Management Coalition – The Workflow Reference Model. Technical report, Workflow Management Coalition, Lighthouse Point, Fla., January 1995. Technical Report WFMC-TC00-1003, version 1.1.

[24] Ning Hu, Randy K. Smith, and Phillip G. Bradford. Security for fixed sensor networks. In *Proceedings of the 42nd annual Southeast regional conference*, pages 212–213. ACM Press, 2004.

[25] IDI. IDI website. http://www.idi.ntnu.no, Accessed October 7th 2005.

[26] IEEE. IEEE Xplore portal. http://ieeexplore.ieee.org/Xplore/DynWel.jsp, Accessed June 7th 2005.

[27] Kristoffer Jacobsen. Organizing mobile work processes in pervasive computing environments, master thesis. Technical report, 2005.

[28] Peter J. Kammer, Gregory Alan Bolcer, Richard N. Taylor, Arthur S. Hitomi, and Mark Bergman. Techniques for Supporting Dynamic and Adaptive Workflow. *Computer Supported Cooperative Work*, 9(3/4):269–292, 2000.

[29] Aker Kværner. Aker verdal. http://www.akerkvaerner.com/Internet/AboutUs/GroupStructure/FieldDevelopmentEurope/AkerVerdal.htm.

[30] Kalle Lyytinen and Youngjin Yoo. Issues and Challenges in Ubiquitous Computing. *Communications of the ACM*, 45(12):62–65, December 2002.

[31] Peter Mangan and Shazia Sadiq. On building workflow models for flexible processes. In *CRPITS '02: Proceedings of the thirteenth Australasian conference on Database technologies*, pages 103–109, Darlinghurst, Australia, Australia, 2002. Australian Computer Society, Inc.

[32] Peter J. Mangan and Shazia W. Sadiq. A constraint specification aproach to building flexible workflows. volume 35, pages 21–39, 2003.

[33] Microsoft. MS Embedded Visual C++. http://msdn.microsoft.com/mobility/othertech/eVisualc/default.aspx, Accessed October 18th 2004.

[34] MOWAHS. MObile Work Across Heterogeneous Systems. Web: http://www.mowahs.com, 2004.

[35] Bonnie A. Nardi, editor. *Context and Consciousness: Activity Theory and Human-Computer Interaction*, pages 69–102. MIT Press, 1996.

[36] Bonnie A. Nardi, editor. *Context and Consciousness: Activity Theory and Human-Computer Interaction*, pages 73–76. MIT Press, 1996.

[37] Bonnie A. Nardi, editor. *Context and Consciousness: Activity Theory and Human-Computer Interaction*, pages 71–73. MIT Press, 1996.

[38] NTNU. NTNU website. http://www.ntnu.no, Accessed June 6th 2005.

[39] Jon Ole Nødtvedt and Man Hoang Nguyen. Mobility and context-awareness in workflow systems. Technical report, Dep. of Computer and Information Science, NTNU, Norway, June 2004. Diploma thesis (174 p.).

[40] Computer Graphics Technology Group of PUC-Rio. The programming language lua. http://www.lua.org/.

[41] David L. Parnas. *On the criteria to be used in decomposing systems into modules*, pages 411–427. Springer-Verlag New York, Inc., New York, NY, USA, 2002.

[42] Thomas F. La Porta, Krishan K. Sabnani, and Richard D. Gitlin. Challenges for Nomadic Computing: Mobility Management and Wireless Communications. *Mobile Networks and Applications*, 1(1):3–16, 1996.

[43] Gary Riley. CLIPS – A Tool for Building Expert Systems. http://www.ghg.net/clips/CLIPS.html, 2003.

[44] Kay Römer, Oliver Kasten, and Friedemann Mattern. Middleware Challenges for Wireless Sensor Networks. *ACM SIGMOBILE Mobile Computing and Communications Review*, 6(4):59–61, 2002.

[45] M. Satyanarayanan. Pervasive Computing: Vision and Challenges. *IEEE Personal Communications*, 8(4):10–17, August 2001.

[46] M. Satyanarayanan. Challenges in Implementing a Context-Aware System. *IEEE Pervasive Computing*, 1(3):2–2, July-September 2002.

[47] M. Satyanarayanan. Of Smart Dust and Brilliant Rocks. *IEEE Pervasive Computing*, 2(4):2–4, October-December 2003.

[48] M. Satyanarayanan. Privacy: The Achilles Heel of Pervasive Computing? *IEEE Pervasive Computing*, 2(1):2–3, January-March 2003.

[49] Cite Seer. Cite Seer portal. http://citeseer.ist.psu.edu/, Accessed June 7th 2005.

[50] Springer. Springer portal. http://springerlink.metapress.com/, Accessed June 7th 2005.

[51] Martin Strohbach, Hans Gellersen, Gerd Kortuem, and Christian Kray. Intelligent Artefacts: An Embedded Systems Approach for Cooperative Assessment of Situations in the World. In *The Sixth International Conference on Ubiquitous Computing (Ubicomp 2004)*, Nottingham, England, September 7-10 2004.

[52] Sun Microsystems, Inc. Model-view-controller. http://java.sun.com/blueprints/patterns/MVC-detailed.html.

[53] Carl-Fredrik Sørensen, Alf Inge Wang, and Reidar Conradi. Support of smart work processes in context rich environments. Submitted to MOBIS'2005, Leeds UK.

[54] The IEEE Working Group for WLAN Standards. IEEE 802.11 Wireless Local Area Networks. http://grouper.ieee.org/groups/802/11/, 2002.

[55] The Workflow Management Coalition. Workflow Process Definition Interface – XML Process Definition Language, 2002.

[56] The Workflow Management Coalition. Welcome to WFMC. http://www.wfmc.org, 2003.

[57] Universidade de Lisboa, Lancaster University, Trinity College Dublin and Universität Ulm. CORTEX project homepage. http://cortex.di.fc.ul.pt, 2001.

[58] Earth System Science Center University of Alabama in Huntsville. Sensor Modeling Language (SensorML). http://vast.uah.edu/SpaceTimeToolkit, Accessed June 7th 2005.

[59] Earth System Science Center University of Alabama in Huntsville. University of Alabama in Huntsville, Earth System Science Center Home Page. http://vast.nsstc.uah.edu/SensorML/, Accessed June 7th 2005.

[60] Mark Weiser. The Computer for the 21st Century. *IEEE Pervasive Computing*, 1(1):18–25, January-March 2002. Reprinted from Scientific American, 1991.

[61] Wondermar. Shipyard scenario. http://www.wondermar.net/, Accessed October 18th 2004.

[62] World Wide Web Consortium – W3C. Extensible Markup Language (XML). http://www.w3.org/XML/, 2002.

[63] Maomao Wu, Adrian Friday, Gordon Blair, Thirunavukkarasu Sivaharan, Paul Okanda, Hector Duran Limon, Carl-Fredrik Sørensen, Gregory Biegel, and René Meier. Novel Component Middleware for Building Dependable Sentient Computing Applications. In *Proceedings of the ECOOP'04 Workshop: Component-Oriented Approaches to Context-Aware Systems*, Oslo, Norway, June 14th 2004.

[64] Marvin V. Zelkowitz and Dolores R. Wallace. Experimental Models for Validating Technology. *Computer*, 31(5):23–31, May 1998.