NTNU
Innovation and Creativity

Arbitration and Planning of Workflow Processes in a Context-Rich Cooperative Environment

Christian Indahl
Kjell Martin Rud

Master of Science in Computer Science

Submission date:	June 2007
Supervisor:	Alf Inge Wang, IDI
Co-supervisor:	Carl-Fredrik Sørensen, IDI

Norwegian University of Science and Technology
Department of Computer and Information Science

Problem Description

We will look at mechanisms that are needed to automatically construct workflows, with only a desired goal as a starting point, and which technologies that can support automatic workflow construction. In addition, we would like to take a closer look at how workflows can be made adaptive based on context and how to ensure minimal replanning in schedules.

We also want to investigate how to represent a conflict between two clients in a dynamic and generic way in a cased-based reasoning repository. The conflicts must be recognisable in a different context and the CBR framework must be able to reason around the nature of the conflict so it can detect unknown conflicts that are similar.

Assignment given: 20. January 2007
Supervisor: Alf Inge Wang, IDI

Abstract

Hardware has come a long way to support pervasive computing and workflow management, whilst software has fallen behind. Existing systems lack the ability to make decisions that corresponds with user intents and are unable to handle complex context-rich workflow conflicts.

Since workflow systems are meant to facilitate normal workers, we have looked at how workflows can be generated and adapted without prior knowledge of programming. The approach is based on the elaboration of so called *calm technologies*, bringing user interference to a minimum. We propose ways of automating the process of obtaining context, generating workflows, making plans, and schedule resources before execution.

Obtaining context is proposed done by a *Context service* which delivers tailored context information through abstraction. To create workflows, the only thing a user needs to know is what he wants to achieve. The rest is generated. The planning mechanism used is the *Scheduling service* first proposed in our depth study [75]. As a part of this, we describe a method for how to simulate future context for better planning purposes, decreasing the need for adaption and replanning caused by context changes.

When several actors execute workflows in an environment, conflicts will occur. We have made a proof-of-concept implementation of the *Arbitration architecture* from our depth study. This approach used case-based reasoning to recognise conflicts between workflows and select a solution. We set out to find a way to store a conflict representation as a CBR-case so it can be recognised in a different context and enable the service to recognise conflicts that are similar in nature.

We found that a case could be stored using ontologies to describe the nature of the workflow constituents that make up the conflict. In addition, *context* and *state triggers* are proposed. These filter the cases that can not be conflicts, due to current contextual information or other states, before the CBR framework computes similarity of the cases against the current workflows. Using an *expert system* supporting *fuzzy logic*, it could speed up the similarity computations required to recognise conflicts. After running some scenarios, we found that the system was able to detect known conflicts in a different context and discover prior unknown. This was achieved because of the similarity in nature to a known conflict.

Preface

This report is written as a master thesis on context-aware workflow enactment systems at the Institute for Computer and Information Science (IDI) at the Norwegian University of Science and Technology in Trondheim.

The authors of the report are *Christian Indahl* and *Kjell Martin Rud*, with the project running from mid January to mid June 2007.

We would like to thank our supervisors, PhD Carl-Fredrik Sørensen and PhD Anders Kofod-Petersen, for excellent support, guidance and comments.

Trondheim, 14. June 2007

Christian Indahl *Kjell Martin Rud*

Contents

List of Tables

List of Figures

PART I

Context

CHAPTER 1

Introduction

1.1 Background

The work presented in this master thesis is a continuation of our depth study [75], which was based on work within the research project *Mobile work across heterogeneous systems* (MOWAHS). This was a basic research project supported by the Norwegian Research Council in its IKT-2010 program, and was carried out jointly by the Software Engineering and Database Technology groups at NTNU. The project had two main parts: process support for mobile users using mobile devices, and support for transactions/workspaces incorporating work documents [98], where our contribution relates to the first part.

The MOWAHS project has led to several papers, reports, software, Web material, and PhD theses. The contributions with most relevance to our work are those related to *Smart work processes* (Section 6.1), a concept developed by Sørensen et al. [88, 89]. This is a combination of ubiquitous computing and workflow that defines a new research direction to be investigated.

In our depth study [75], we addressed some of the challenges indicated in [89]. In this master thesis we will explore these solutions further, and develop a proof-of-concept prototype to gain better insight and find out if our findings are feasible.

1.2 Motivation

The overall motivation for our work is to make a contribution to the ubiquitous computing domain by addressing some of its problem areas. The research focus of this master thesis is indicated by the following motivations.

1.2.1 Automate activities regardless of time and location

Humans travel more and more, both privately and work related. The spread of mobile phones have made people accessible regardless of where they are. As mobile phones evolve into more powerful handheld devices, new and more resource-demanding services can be made available. This, in combination with the development of better sensor and actuator technology, could be a step towards *Weiser's vision* [99]. New mobile and wireless technology offers the opportunity to automate and support real-time, in situ work processes, hence supporting automation of both everyday trivial tasks as well as more complicated work related tasks.

1.2.2 Support a dynamic work environment

The real world is a dynamic construct. People constantly act in accordance with actions invoked by other people and the environment itself. As a result, work is often performed ad-hoc, requiring adaptation to the situation at hand. This dynamic process requires communication, cooperation, and conflict resolution amongst several persons. This is a fairly new and complex research field where good computer solution are sought after.

1.2.3 Prove concepts from our depth study

The architectures provided in our depth study, *arbitration* and *scheduling* [75], constitute important ideas and foundational services in an ubiquitous environment. These architectures need to be developed and tested. In this process, new problems will arise and generate new questions which will help us gain better insight and propose better solutions.

1.3 Problem definition

An increasing number of sensors and actuators resides in our daily environment. In the future we envision that these are readily available for use by portable devices. These devices can then control or help out with tasks by enacting workflows (Section 5.5.4).

Conflicts between devices running workflows are inevitable. Conflicts must be detected, preferably in advance, and solved. We will create a proof-of-concept implementation of the arbitration architecture from our prestudy [75], and draw experiences from this. Our main focus will be on how to discover conflicts rather than how to solve them.

Workflows are written in languages closer to programming than natural language. This renders most people unable to create new workflows from scratch. They need to be put together in a user friendly fashion, preferably by only knowing what the user want to achieve, or by providing ready-to-use building blocks.

Section 2.1, Research questions, describes in more detail the problems we set out to solve.

1.4 Limitation of scope

There are many important challenges in the research field of ubiquitous computing. This makes it necessary for us to abstract our work away from the areas which do not concern our research. In addition, assumptions have to be made, making it possible to generate solutions for the future, without going into great detail on present problems.

The following states what research areas we will not look into, in order to focus only on our restricted subjects.

Communication systems The field of communication topology will be briefly covered, but we will not touch terms like roaming profiles of mobile systems, connection coverage, or bandwidth limitations.

Security Some people would most likely try to take advantage of a smart workflow system by feeding it with incorrect data, stealing identities, or just trying to prevent communication to take place.

We will not address any security matters related to the authenticity of sensors, actuators, mobile workers, or clients. Neither will we handle the correctness or integrity of data.

Pervasive environment The *Smart work processes* architecture (Figure 6.1), has its own handler for communicating with sensors and actuators. We handle an actuator as a single actuator even though it may be a compound of actuators with a set of internal reliance's. E.g., a compound actuator may be several heaters working together to raise the temperature of the room. Each can be controlled individually since they might need different appliance of power to increase or hold the temperature stable across the room. A compound actuator may be a set of heaters where you only set the temperature for the room all together and this results in an diverse request to the different heaters.

System implementation and deployment Development and testing will happen only in a simulated environment on the computer. A real world application will not be prioritised.

User intent Inference of user intent will not be explored. We will rather provide this explicitly through goal description of each activity and possibly through dynamic interactions with the user.

1.5 Assumptions

Some parts of the environment are to dynamic to take into consideration in this master project. We therefore make several assumptions to ease the work on our main objective.

Computational power As handhelds grow ever stronger there is a possibility for tomorrow's handhelds to compute large amount of data. In our study we assume that handhelds have this computational power or forward computations to other more powerful computers.

Memory and storage constraints Although mobile devices today are constrained by memory and storage capacity, we assume that tomorrow's devices have enough of these resources to accommodate the scheduler and arbitrator.

Energy consumption Mobile devices are gradually required to execute more advanced applications and interact with others using wireless communication technology. This requires that energy sources must become more powerful, and that the energy consumption, in general, must be reduced. We assume that future devices have enough energy for mobile collaboration.

Screens Screens grow ever better with respect to the visible size and resolution ratio. It is therefore assumed that tomorrows screens will manage to accommodate large amounts of data, possibly structured in a more elegant way.

1.6 Outline of the report

Part I - Context

This part gives an introduction to the report by touching subjects that will be described more in detail in the following parts. Our *research area* is presented through the motivation, problem definition, and research questions. Then, our *approach* is described by a set of research methods. The proposed answers to our research questions and an evaluation of the research methods are included here, as well as in the discussion (Chapter 16).

Part II - Prestudy

Part II, consists mainly of our literature survey, and provides a top-down approach to the theoretical background and state-of-the-art for our *research area*.

We begin with the evolution of computer science within our context, before we move on to issues in ubiquitous computing. We then explore a set of theoretical methodologies, supporting technologies and terms we find interesting and relevant for promoting our research questions. Finally, we present some frameworks and middleware that relates to theory and were considered for our proof-of-concept implementation.

Part III - Arbitration

This part handles our contribution related to arbitration in the ubiquitous domain. *RQ1*, with sub-questions, is elaborated in this part.

We start of by presenting the contribution from our depth study [75]. Then we present some scenarios that will help us understand the domain and extract requirements. We then give an architectural description of the *Arbitration service* based on our dept study [75] and the elaborated requirements. The implementation of our proof-of-concept application is then described.

Part IV - Planning

The second research question, *RQ2* with sub-questions, is elaborated in this part. Here, we present a scenario that is intended to help people understand the problem domain. Then we present our work on *workflow generation*. This is then put into context with a *Scheduling service*, which was first presented in our depth study [75].

Part V - Discussion and conclusion

In Part V, we summarise the report by discussing our proposed solutions, how they cover our *research questions*, and what parts of our literature survey we have chosen to use, how.

We then present our *concluding remarks*. After this, we provide a chapter containing what we believe needs to be done in the future regarding our findings, to sustain a progress.

Part VI - Appendix

The appendix, Part VI, contains a more detailed description of some of the document type definitions and XML examples from our proof-of-concept implementation and a small code example.

Last, a glossary and the bibliography is presented.

CHAPTER 2

Research

2.1 Questions and motivation

According to the *analytical research* method (RM3 in Section 2.2), research should start off with the establishment of research questions. In the process of answering these questions, we end up adding own thoughts and conclusions to the academic discussion.

However, we have started off by using the *literature survey* method (RM1 in Section 2.2), to gain an understanding of the problem domain and the difficulties faced by previous research projects (Part II). This method has helped us perform an improved analytical study.

Based on the analytical study and experience from our depth study, we have established the following research questions relating to our problem definition:

RQ1 *How can we represent a conflict between two clients in a dynamic and generic way in a Case-based Reasoning case repository, so it can:*

RQ1.1 *Recognise the same conflict in a different context?*
Conflicts will arise when two or more actors want to use the same resources to perform work, or if they have contradictory goals. We want to learn from conflicts that arise in order to reuse the knowledge (about how they were solved) in later conflicts. For this to be useful, we must find a way to recognise prior encountered conflicts which arise in a different situation (different context).

RQ1.2 *Predict a possible conflict that is similar in nature to ones that are known?*
The best way to solve conflicts, is to avoid them. Before we can avoid conflicts, we must be capable of recognising them based on obtained knowledge. It is essential to find as many potential conflicts as possible in order to provide an efficient system, therefore we want to recognise conflicts that are not necessary equal, but similar to already discovered conflicts. Supporting this, the system is given the means to learn from prior cases.

RQ2 *How can we support in situ planning of work processes with minimal user interference?*

RQ2.1 *What mechanisms are needed to automatically construct workflows with only the desired goal as a starting point?*
An average user will not know how to programmatically create workflows from scratch. The system must be user friendly, which means that the user should do as little as possible. When wanting to initiate work, we want the user to be able to only describe what he want to achieve. The actual construction of work processes and execution of work must then be automated.

RQ2.2 *How can existing technologies be combined to support automatic workflow construction?*
If we are going to be able to provide a solution for *RQ2.1*, we must figure out which existing technologies to use, and how to use them in combination for this purpose.

RQ2.3 *How can workflows be made adaptive based on context?*
A user might be in a situation which requires that work is done in a special way, taking considerations to the environment. This situation might be temporal, and as it changes, the ongoing work must perhaps be conducted in a different way. We want to find a way where the work description is altered and adapted to the new situation without the user needing to interfere.

RQ2.4 *How to ensure minimal replanning?*
If the user has several things he want to achieve, a plan for how to proceed are needed. After planning has taken place, the situation might change so that the plan is no longer feasible. We want a way to predict what is going to happen with the user's situation in the future, so that this can be accounted for when making the plan, hence reducing the need to make new plans.

2.2 Methods

The research and implementation work is performed in a top-down[1]/ bottom-up[2] fashion, with the purpose of finding a suitable convergence point balancing theory and implementation.

Development of the prototype follows an incremental, service-oriented approach. First we implement basic functionality, and subsequently add more advanced features.

We reflect the prototype development by describing methodologies, technologies, frameworks, and middleware used, including evaluations of these, comments on the choices, difficulties, trade-offs, and results discovered during implementation. The implemented services are also described, with details on how they work and how they relate to the problem definition and given scenarios.

[1]A top-down approach emphasise planning and a complete understanding of the system.
[2]A bottom-up approach emphasises coding and early testing.

In the following, a more detailed description of the research methods is given:

RM1 *Literature survey*
A literature survey is a detailed study of literature, describing already relevant information on the subject. The literature survey gives us an understanding of the problem domain and the difficulties of earlier projects, and helps us perform a more directed, descriptive, and analytical study. It also provides more sensible scenarios and better requirements.

RM2 *Descriptive study*
This method tries to answer questions concerning the current state-of-the-art of the study's subject. It does so without manipulation of variables and does not try to establish causal relationship between events, but simply describes them [59].

Adopted basic steps of descriptive research based on [74]:

- recognizing and identifying a topic to be studied
- selecting appropriate scenarios to describe the area
- collecting reliable and valid data
- reporting conclusions

A descriptive method may be used to gain more information about a subject and use this to generate theories [37].

RM3 *Analytical study*
Analyse means to break a topic or concept down into its parts in order to inspect and understand it, and to restructure those parts in a way that makes sense to the researcher. In analytical research, research is done to become an expert on a topic so that the parts of the topic can be restructured and presented from the researches own perspective [70].

With this type of research, we start off with research questions. We then try to answer these questions by studying information and views with critical thinking and reading, plus evaluation of the resources. In this way, we end up by adding our own thoughts and conclusions to the academic discussion.

If the study aims to actually test preplanned theories based on existing knowledge or findings, an analytical research method will be needed [37].

RM4 *Scenario building*
Scenarios are characterisations of users and their tasks in a specified context. They offer concrete representations of a user working with a computer system to achieve a particular or several goals. These scenarios can be created by the development team, stakeholders, or the target users, and are intended to provide the developer with usability requirements [24].

[24] also states certain benefits with scenario building:

- Scenario building encourages designers to consider the characteristics of the intended users, their tasks, and their environment.
- Usability issues can be explored at a very early stage in the design process (before a commitment to code has been made).
- Scenarios can help identify usability targets and likely task completion times.

- The method promotes developer involvement and encourages a user-centred design approach.
- Scenarios can also be used to generate contexts for evaluation studies.
- Only minimal resources are required to generate scenarios.

Adopted principle steps for this method are as follows:

- Gather together the research team and other relevant stakeholders under the direction of an experienced facilitator.
- Identify intended users, their tasks, and the general context. This information will provide the basis for the scenarios to be created by the research team.
- Functionally decompose user goals into the operations needed to achieve them.
- Assign task time estimates and completion criteria as usability targets

We provide scenarios that describe how a system should or could work with humans.

RM5 *Requirements elicitation - Greenfield Engineering*
With visionary scenarios, Greenfield engineering is often used [4]. This is because Greenfield engineering is a method of requirement elicitation that takes into account that development starts from scratch, no prior system exists and the requirements are extracted from the end user, scenarios, and prior projects [69, 15, 4].

In Figure 2.1, we describe how the research methods assist us in different aspects of the project and how they relate.

2.3 Propositions to the research questions

RQ1

How can we represent a conflict between two clients in a dynamic and generic way in a Case-based Reasoning case repository, so it can:

RQ1.1

Recognize the same conflict in a different context?

Storing a representation of a situation is not too difficult. The problem lies within filtering out the important information and storing it. This should not only be done to save storage space, but mostly because some contextual information is not important and therefore can hinder the situation from being recognized. E.g., you want to recognise a *control temperature* conflict regardless of a chair's position in the room.

The selection of these properties, or contextual elements is subject to constant research and is not a part of our study. We only see how this information can be stored in a case repository when it has been identified.

Our solution is to represent the nature of the conflicting workflows or activities as ontologies in the case-base. By doing this we are able to predict conflicts in workflows or activities of similar nature.

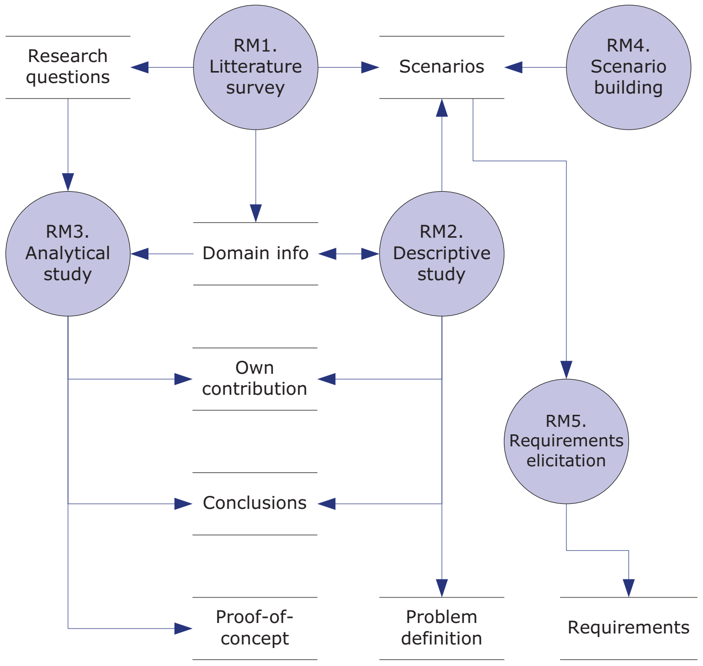


Figure 2.1: How the described research methods are used to support different aspects of the project.

In the case-base, the contextual states that identify a conflict are stored as a tree. These are called contextual or state triggers, as they may check for more than just context. Triggers must contain the general domain knowledge of the context entity so they can reason for interchangeable context entities and the state that must be satisfied.

An *expert system* should sit between the case repository and the main CBR cycle. It should check contextual and state triggers and filter the list of cases to achieve efficiency. The computer-intensive process of CBR similarity-computation could be limited to a minimum by being used in combination with the very cost effective expert system.

RQ1.2

Predict a possible conflict that is similar in nature to ones that are known?

The proposed answer to this question also include the use of ontologies. The workflows' or activities' classifications that make up the conflict description depict their nature. Classifications in proximity should encounter the same difficulties in concurrent and cooperative situations.

As a result of the classifications, it is possible to derive and possible foresee problems that are not composed of the same workflows or activities as the ones known. Using this technique, the system is able to learn a new unknown conflict by itself, adapt, and invoke a solution.

There is a great challenge to this approach. That is to keep the general domain knowledge consistent, correct and maintain the semantic quality in a network where changes can be made by anyone.

RQ2

How can we support in situ planning of work processes with minimal user interference?

In the quest of a system which requires a minimum of interference from the user when planning and executing work, a technique must be elaborated that can automate this based on available knowledge and information.

Our prerequisites in the elaboration of this question are that workflows are made up by a chain of activities, the motivation for a workflow is to achieve a goal, and that workflows are the constituent parts of a plan. We derive the following subquestions:

RQ2.1

What mechanisms are needed to automatically construct workflows with only the desired goal as a starting point?

Basic criteria for being able to address this question is the presence of contextual information to base decisions upon, and a method to obtain such information. The requirement of minimal user interference indicates that activities must be obtained from some sort of knowledge or experience base. An activity must then have certain attributes that makes it possible to filter relevant activities from those not suited for achieving the goal.

We believe that one attribute should contain information of contextual requirements that must be satisfied before the activity can be executed. Another attribute should be success criteria, which give us an understanding of what the activity aims to achieve. Provided fidelity is also be relevant where quality is important for fulfilling the goal. As activities are reusable assets, it is also of interest to know about how well an activity accomplish its function.

With access to this information, and available activities, we have the basics for finding and adapting the activities that best suits the contextual situation and results in achievement of the workflow's goal.

RQ2.2

How can existing technologies be combined to support automatic workflow construction?

Gathering contextual information is proposed done by a *Context service* (Section 14.1), which is based on already existing research.

To be able to provide the mentioned attributes in an activity description, we have ended up with a structure based on the *Resource description framework* (Section 5.3.1 and 5.3.2). This language supports *ontologies* (Section 5.3.2), which allows measuring of hierarchical relationships with other activities.

How an activity has actually performed and what results it has delivered, can be measured by using *Reinforcement learning* (Section 5.4.5). This information should probably be included into the activity description along with the other attributes. Such information could then be used in finding the best suitable activity amongst similar activities.

The actual proposition for finding suitable activities when assembling a workflow is based on *Case-based reasoning*. By saving experience about which activities best satisfies what goal, makes us capable of reusing whole workflows. If no complete workflows exists that satisfy the goal, it is possible to construct a workflow, based on activitie's attribute values, by choosing activities that constitutes the difference in current situation and the goal's desired situation.

RQ2.3

How can workflows be made adaptive based on context?

Situated planning as described by Bardram [12], is a consequence of actions performed *in situ* which changes the contextual conditions.

To keep a workflow situated (executable in the current situation) while context changes, it needs to be adaptive. An adaptive workflow must be able to exchange one or more of the activities that represents its constituent parts, so that constraints are satisfied.

By using the mechanisms and technology refered to in *RQ2.1* and *RQ2.2*, it should be possible to obtain better suited activities and insert these to the relevant workflow. However, it is not desirable with workflow adaption whenever a contextual parameter changes. To make workflows more robust, we suggest the use of *Expert systems* (Section 5.4.2) with *Fuzzy logic* (Section 5.4.4) when interpreting an activity's constraints and success criteria.

However, exceptions in the process will occur from time to time because of contextual change. For obtaining substitute activities we suggest *case-based reasoning* or *soft case-based reasoning* which have better capabilities and include *fuzzy logic* amongst other techniques.

RQ2.4

How to ensure minimal replanning?

Replanning takes place when the current plan cannot execute due to context conditions. Reordering and finding combinations of workflows that works, or re-constructing the relevant workflows are then alternatives for getting a functional plan.

However, such replanning is resource demanding, and should therefore be kept at a minimum. We suggest an alternative approach, the *Scheduling service*, where contextual state are simulated based on activities' assumed influence on context (Chapter 15). Information of an activity's context influence should also be in its description, as stipulated in *RQ2.1*. The creation and improvement of such information could be taken care of by *Artificial neural nets* (Section 5.4.6) and *Reinforcement learning* (Section 5.4.5).

Another means to achieve less replanning is to avoid conflicts. If a conflict occur amongst actors, replanning is most likely an outcome for one or more parties. One way of preventing conflicts, is to share schedules with other entities within the perimeter of the relevant environment.. By doing this, other planning services can take these schedules into account when planning. With many actors within an area, this could however lead to massive distribution of schedules, and diminish the value of the approach. Letting resources keep an up to date schedule for when they are available and when they are booked by some entity, seems like a more reasonable approach.

When conflicts arise, they need to be solved. The negotiation process is conducted by the proposed and implemented *Arbitration service* (Part III). How this solves conflicts will also affect the need of replanning.

2.4 Methods evaluation

Here we will evaluate our research methods. Figure 2.1 shows how the different methods have been used during our project.

RM1 *Literature survey*

We have used *RM1* to read up on theories and technologies that will help us understand the problem domain. Doing this, we have gained valuable knowledge that has helped us construct our research questions and thereby also helped us form our scenarios. During our project, this method has been used several times to gain knowledge on newly discovered topics.

RM2 *Descriptive study*

This method has been used to go deeper into the problem domain and develop it further. When selecting our scenarios, we used *RM2* togheter with the results of *RM1* to identify what key points the scenario had to contain.

Using a descriptive study, we grew more familiar with the state-of-the-art from our researchfield. This helped us form our problem definiton, which again was a guidance for our research questions.

This method also directly contributed to the making of our own contribution and conclusion.

RM3 *Analytical study*

Having obtained a deeper knowledge of the domain in question and formed research questions, *RM3* was used to break our findings into more understandable pieces. We then restructured these and added our own thoughts to the work.

During the implementation phase, *analytical study* contributed to the decisions that were taken.

RM4 *Scenario building*

We have created scenarios to help the readers, and our selves, understand how the system would ideally work and the problems that have to be overcome.

The scenarios were the foundation for the planning, architecural design, requirements and development of our proof-of-concept implementation.

RM5 *Requirements elicitation – Greenfield engineering*

RM5 was used to extract requirements from the constructed scenarios and the prior work from our prestudy [75].

PART II

Prestudy

CHAPTER 3

Evolution of computer science

Ubiquitous computing, first visioned by Weiser [99], is a complex research area with many hurdles to overcome. In [77], Satyanarayanan describes a categorisation of computer system issues existing before and after the introduction of ubiquitous computing. In the following, we use this to gain a better overview and understanding of the ubiquitous computing domain.

3.1 Distributed systems

The field of distributed systems has created a conceptual framework and algorithmic base that has proven to be of enduring value in all work involving two or more computers connected by any kind of network. This body of knowledge spans many areas that are foundational to pervasive computing:

- *Remote communication* (protocol layering, RPC, timeouts, and end-to-end arguments)
- *Fault tolerance* (atomic transactions, distributed and nested transactions, and two-phase commit)
- *High availability* (optimistic and pessimistic replica control, mirrored execution, and optimistic recovery)
- *Remote information access* (caching, function shipping, distributed file systems, and distributed databases)
- *Security* (encryption-based mutual authentication and privacy)

3.2 Mobile computing

A distributed system with mobile clients constitutes mobile computing, which is the next step in evolution. Many of the basic principles of distributed system design apply, but four key constraints of mobility have forced the development of specialised techniques:

- Unpredictable variation in network quality
- Lowered trust and robustness of mobile elements
- Limitations on local resources imposed by weight and size constraints
- Concern for battery power consumption

Mobile computing's growing knowledge base spans the following broad areas:

- *Mobile networking* (mobile IP, ad-hoc protocols, and improved TCP performance in wireless networks)
- *Mobile information access* (disconnected operation, bandwidth-adaptive file access, and selective control of data consistency)
- *Support for adaptive applications* (transcoding by proxies and adaptive resource management)
- *System-level energy saving techniques* (energy aware adaption, variable-speed processor scheduling, and energy-sensitive memory management)
- *Location sensitivity* (location sensing and location-aware system behaviour)

Sørensen et al. [88, 89] differentiate *mobile work* from *nomadic work*, where *nomadic work* refers to anywhere, anytime computing which is not regarded as context-dependent. *Mobile work* is performed in a mobile environment dependent of context information extracted from the physical environment. That is, mobility is necessary to accomplish the process goals. Mobile work and the local working environment can computationally and physically mutually influence each other by observed values from sensors or changes made by actuators.

3.3 Ubiquitous and pervasive computing

Weiser's vision [99] foresee that computers and humans will seamlessly interact in such a way that technology becomes invisible to us. Weiser calls this *Ubiquitous Computing* which is a reflection on what could happen when computers are so small and cheap that they fade into the background. Satyanarayanan later introduce the notion of *Pervasive Computing* [77].

Lyytinen [64] presents an overview of the dimensions of Ubiquitous Computing (Figure 3.1). This can be used to separate the Ubiquitous and Pervasive paradigms.

This figure presents the dimensions on making the computer invisible. Lyytinen proposes that the main challenge in relation to Ubiquitous Computing is the integration of large scale mobility with Pervasive Computing functionality [48].

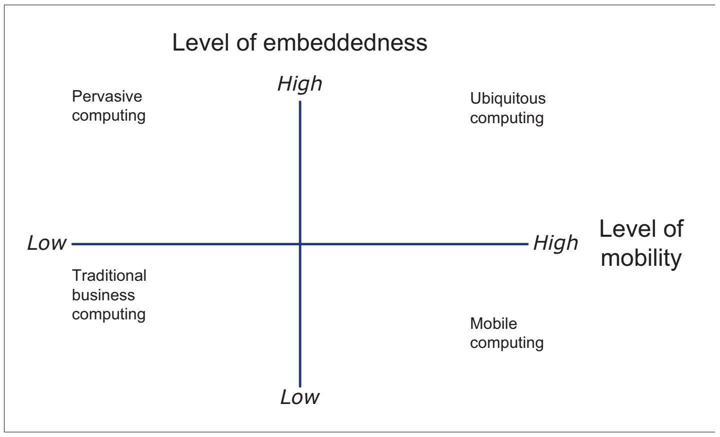


Figure 3.1: Overview of the Ubiquitous Computing dimensions.

Petersen and Kofod-Petersen [72] deals with Ubiquitous and Pervasive Computing in order to describe *Ambient intelligence*, which they see as a combination of a number of paradigms; Ubiquitous Computing [99], Pervasive Computing [77], and Artificial Intelligence [76]. Figure 3.2 shows the relations between these paradigms. The Ubiquitous Computing aspect addresses the notion of accessibility of the technology, where the technology and connectivity is available through everyday objects that are in the user's environment. Artificial Intelligence techniques provide the context awareness to establish the user's needs and the appropriate response. The Pervasive Computing aspect supports the architectural aspects to realise the situation.

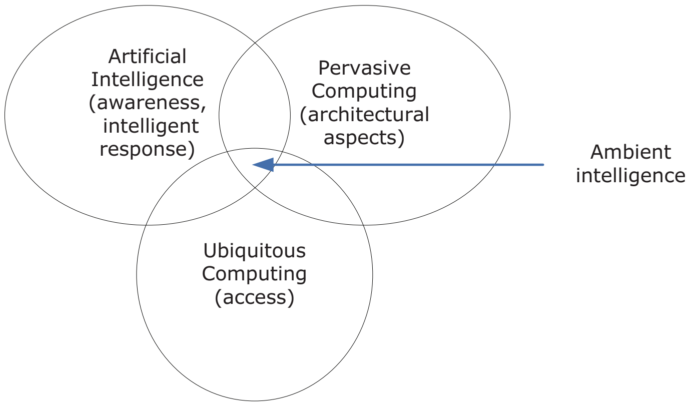

Artificial Intelligence (awareness, intelligent response)
Pervasive Computing (architectural aspects)
Ambient intelligence
Ubiquitous Computing (access)

Figure 3.2: Ambient intelligence paradigms (based on [72]).

Despite the differences in the Ubiquitous Computing and Pervasive Computing paradigms, we will treat the terms as synonyms in this report, in accordance with Satyanarayanan [77].

The research agenda of Pervasive Computing subsumes that of mobile computing, but goes much further. Specifically, Pervasive Computing incorporates four additional research subjects into its agenda [77]:

Effective use of smart spaces:
A space may be an enclosed area or a well-defined open area. The fusion of computing and spatial infrastructure enables sensing and control of one world by the other. A simple example of this is the automatic adjustment of heating, cooling, and lighting levels in a room based on an occupant's electronic profile. Software on a user's computer may also behave differently depending on where the user is currently located.

Visibility of technology:
The ideal expressed by Weiser's vision [99] is complete disappearance of technology from a user's consciousness. In practice, a reasonable approximation to this ideal is minimal user distraction. This can be achieved if a pervasive computing environment continuously meets user expectations and rarely present surprises.

Localized scalability:
As smart spaces grow in sophistication, the intensity of interactions with the surroundings increases. The presence of multiple users will further complicate this problem. As a means to limit this problem, [77] propose that the density of interactions between peers has to fall off as distance increases between them. If no such action is taken, both the user and the computing system can be overwhelmed by distant interactions that may be of less relevance than the in situ interactions.

Masking uneven conditions:
Widespread adoption of ubiquitous computing, if it is ever achieved, is probably many years or decades away. In the meantime, there will persist huge differences in the "smartness" of different environments because of the uneven deployment. One way to reduce the amount of variation seen by a user is to have his/her personal computing space compensate for "dumb" environments by, e.g., letting the system being capable of disconnected operation.

CHAPTER 4

Issues in the ubiquitous domain

Many of the hardware and software technologies necessary to realise ubiquitous computing exist and are even commercially available. A big part of the research should therefore be focused on seamless integration of these technologies.

We will in this chapter give an overview of domain issues especially prominent from our point of view, and analyse them with the intention of finding suitable techniques for supporting our proof-of-concept implementation and theoretical contribution. Findings will be described briefly here, and more thoroughly in Chapter 5.

4.1 World perception

The demand for invisibility in ubiquitous computing requires minimal user intrusion, through *Calm technologies* [100]. This requires that computers interact and make decisions on our behalf, which makes it interesting to explore how humans percieive the world and what mechanisms make us able to learn from experience. This knowledge can be used to better understand what techniques we should investigate regarding further research.

For computers to be able to react to the environment, they need to be *context-aware* (Section 5.1). However, this is not a straight forward procedure; interpretation of information is dependent on several factors like the observer's knowledge, previous experience, surrounding context, mental and physical shape, interest, stress, and so on. Nevertheless, such information must be caught and interpreted, letting pervasive applications adapt to the changing environment while supporting the user in the best possible way.

Schank and Abelson [80], argue that stories about a person's experiences and the experiences of others are the fundamental constituents of human memory, knowledge, and social communication.

4.1.1 Human memory

Memories are processed and stored in the brain. It is estimated that the brain consists of 180 billion nerve cells, connected through each other through synapses[1]. Each nerve cell has 2000-5000 synapses, which gives the brain an incredible amount of possible ways to interconnect and create patterns. In addition, each of these associations has a dynamic weighting to indicate the strength of each relation. Both cells and synapses may change with age and experience. All this makes memory possible [53].

Information is provided to the nervous system through sensing, and integrated with previous information. A nerve cell may be looked at as a small calculator which receives activating and preventive signals from other cells and performs calculations all the time. When the calculation equals the expected solution, a nerve signal is triggered. Preventive signals reduce probability for the next cell to forward a signal, and activating signals amplifies this probability.

The behaviour presented above is in computer science imitated by *Artificial neural networks* (Section 5.4.6). The broad categories to which artificial neural networks are applied, are classification, function approximation, and data processing.

There are several ways to categorise memories; based on duration, nature of information, and retrieval of information. The most interesting categorisation from our point of view, is by the nature or type of information. The information types found in this category are; declarative (explicit) and procedural (implicit) [107].

Declarative memory requires conscious recall, and can be further sub-divided into semantic memory and episodic memory. Semantic memory concerns facts taken independent of context and allows the encoding of abstract knowledge about the world. In contradiction, episodic memory concerns information specific to a particular context, such as a time and place. Further, episodic memories refer to a semantic representation and all new experiences connected to an activity will modify the single semantic representation of this activity (Figure 4.1).

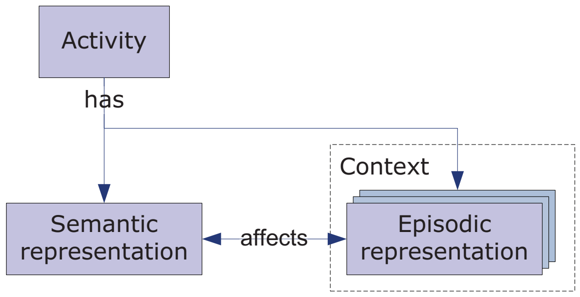


Figure 4.1: Declarative memory of an activity.

The semantic representation is suitable for recognition and classification, while the episodic representation also contains context specific information for each occurrence of the activity. This information is used to update and modify the semantic representation.

Procedural memory is not based on the conscious recall of information, but on implicit

[1]Synapses is small gaps in which chemical signals may cross between nerve cells.

learning. It is revealed when we do better in a given task only because of repetition. No new explicit memories have been formed, but we are unconsciously accessing aspects of those previous experiences.

Categorising memories as described above, reveals that we create abstract representations from context-independent facts about diverse objects, situations, activities, etc. These representations are then continuously used as reference models in the ongoing process of recognising and categorising perceived information. Further, by repeating a specific task in different contexts, the abstract model representing it will become more comprehensive and robust. Repeating a task within the same context increases the performance. In pervasive computing, this could be done by specializing the relevant *Smart space* to support the execution of a particular task.

Mimicking declarative and procedural memory can help us recognise information and automate execution of activities with improved quality. Several techniques exist that support these processes in different ways:

- **Case-based reasoning** (CBR) is the computer science approach to human problem solving. It provides a reasoning mechanism that uses previous episodic knowledge to solve new problems (Section 5.4.1).
- **Fuzzy logic** is used to avoid sharp boundaries between similar concepts. It is well suited for generalising from prototypical cases to new situations (Section 5.4.4).
- **Ontologies** are collections of information that define the relations among terms (Section 5.3.2).

4.1.2 Obtaining knowledge - sensing and abstracting context

Different senses stimulate specialised areas in the brain. This specialisation or categorisation represents a separation of concern which could be considered when structuring and representing environmental information in a pervasive system. Sensor types could be classified by abstractions representing the human senses. By mimicking human perception and categorisation, a more intuitive user and programming interface can be achieved. A technology that supports classification is *ontologies* (Section 5.3.2) which can be represented by the *Resource description framework*.

The abstraction mechanism could be a semantic lookup service like the *Semantic Web* (Section 5.3.1). It could use techniques like ontologies, metadata, taxonomies, thesaurus, and vocabularies to find the proper semantic abstractions.

A reasonable place to localise this kind of sensing service, would be in a *Context service*, as described by Sørensen et al. [88, 89] in their work on *Smart work processes* (Section 6.1). Also, the *Context service* variant implemented by Bardram in [14] (Section 6.3) is an interesting choice.

Wittgenstein, an Austrian philosopher who contributed several ground-breaking ideas to philosophy, observed that concepts that are part of the natural world – such as an orange – are polymorphic [3]. That is, their instances may be categorized in a variety of ways, and it is not possible to come up with a useful classical definition, in terms of a set of necessary and sufficient features, for such concepts. An answer to this problem is to represent a concept extensionally, defined by its set of instances – or cases [3].

Structuring sensed information could in addition be based on the type of sensors that are used to perceive the information. When humans recognise, e.g., an orange, we can use vision or taste as our strongest sense candidates. Smell and feeling can also contribute to recognition, but the utmost advantage would be to use a combination of all of them if available. Information about how objectives are sensed, could be used as a support to using previous experience, and reasoned over by using *Bayesian networks* (Section 5.4.3) to establish a probability of recognition.

4.1.3 Using perceived information

When we walk around in our usual habitat and everything seems familiar, we tend to think little of what is actually surrounding us. In a situation like this, our attention is first triggered if some kind of exception comes into being, i.e., when something is distinctively different than usual. When this happens, we use our senses to collect information about the new situation, which we then analyse to see if it is interesting and if it requires action.

Based on this observation, we see that certain changes in sensor readings should trigger further goal-oriented investigations of the environment. Exactly how to respond to changes in the information flow is activity dependent – does the change effect the execution of the present or planned activities? If so, the procedure must perhaps be changed to reach the activity's goal. This issue is described further in Section 5.5.

When we are challenged by a task that is new to us, it seems like we focus our mental capacity on observing and reason about information that is relevant for solving it. In a similar fashion, peers must collect and reason about information sensed from the environment. What kind of information that should be collected and processed, is again dependent on the task at hand and should be included in the activity description.

By letting an activity only represent a goal, constraints, success criteria, and execution context [89], we can use a technique to recognise what needs to be done in order to fulfil the activity's demands based on previous experience. CBR (Section 5.4.1) is a technique that can handle this approach. This approach requires manually provided information to define new activities, but minimal user interference and dynamic adaption will be ensured as the system's case-base grows bigger.

Because humans have a *habitual memory* [53] (amongst others), tasks we have performed several times will not demand a lot of mental resources because the execution pattern is so well imprinted. This ability relates to Schank's work on dynamic memory [78]; the idea of remembering earlier situations and using situation patterns in reasoning. (This work make some of the foundation of CBR). Much of our mental resources are then available, and the focus can be directed towards something else. This relates to a well imprinted *case* in a CBR system. As activities are repeatedly carried out in different environments under different conditions, the reference case will be more robust (cover more exceptions) and suit more activities with greater quality. However, if any exceptions should arise from the regular pattern, we must relocate our mental focus to solve the new tasks. This phenomena is in cognitive psychology referred to as a *breakdown situation* [97, 54] (Section 5.5.1). The hierarchical structure of activities is in a continuous floating state, and it is worth noticing that, according to [22], *Activity theory* is capable of capturing changing contexts in such break-down situations.

4.2 Resource management

One of the problems when dealing with today's mobile entities is limited resources on the device itself. By this we mean constraints on the device performance, memory capabilities, and energy supply.

Another limitation on resources are those external to a peer, the resources residing in the environment. These resources are mainly network capabilities, sensors, actuators, and other peers which can offer some service. This can also be human resources like a carpenter, plumber, or electrician.

In this section we explore some techniques that support and improve resource management.

4.2.1 Remote execution

As we suggests in our depth study [75], one solution to limited local processing power is *cyber foraging*, a remote execution technique first proposed by Balan et al. [7]. The basic idea of using surrogates to support pervasive computing was introduced in Satyanarayanan's paper on the challenges of pervasive computing [77].

Cyber foraging lets resource-demanding work be executed by more powerful, often centralized peers in the network. When such peers are not available, functionality must be executed on the mobile device. However, when the environment is well-conditioned, resources should be discovered and used. This implies a work-load of some size, but as networks get faster, even small tasks can be distributed for parallel execution. This may not be on powerful, centralized computers, but other mobile peers (see Peer-to-Peer, Section 5.2.1).

Cyber foraging can decrease the storage, battery, and computation requirements of embedded or mobile devices. This, in turn, leads to decreased size, complexity, and cost. Devices would need only enough local resources to perform their common tasks, and could use surrogate resources to perform more complex or less common tasks. A gain from this is the enabling of new and interesting resource demanding applications for mobile devices.

Sachin Goyal and John Carter [45] describe a surrogate infrastructure based on virtual machine technology and show how this reduces the response time and energy drain on the client device. They briefly describe *Spectra*, which is a remote execution environment, built by Flinn et al.[41, 42], that allows a mobile device to use the processing power of a nearby surrogate computer. Spectra monitor application resource usage and the availability of resources in the local environment to decide when and where to utilize cyber foraging. Goyal and Carter take an orthogonal approach to this. They focus on building an infrastructure out of commodity systems and establishing secure surrogate sessions dynamically. Their clients and surrogates do not share a common file system, but instead relies on the surrogate to have an Internet connection to locate and download client application code.

Balan et al. [8] look at data staging and how it is able to provide improved latency for file access without requiring the staging servers to be trusted. They present a remote execution system, *Chroma*, which is able to use extra resources in the environment to improve application performance. They also describe initial work on an environment independent surrogate discovery service.

As *Web 2.0* emerges, we also see that more and more applications and functionality moves from the desktop to the Web interface. This is a form of remote execution that is possible bacause of high speed network technology and a new way of using existing programming technology.

4.2.2 Balanced storing of information

In a ubiquitous environment, it is important that required information is always available where and when needed to support in situ actions. One means to achieve this, is how the information is stored.

Humans carry with them most of the information (knowledge and experience) they need to perform daily activities. Hence, the most prominent information stored in our brains is more or less bound to our usual activities and habitat. If we go away on a vacation, we adapt to the new place and activities through a learning process – our information base is dynamic and adaptive. Much of what we experience is kept in memory for later retrieval and recognition, but because of limited capacity we can not remember every detail.

The brain has to be selective, it must edit and save only what matters. If it did not, it would probably shortcut from all the associations. We therefore need good filters for what gets into memory, and what gets out [53]. The same applies for mobile peers. They have limited resources, and we should be selective of what information is kept and not kept on an entity.

So how should storing of information in a ubiquitous environment be performed? By using humans as the reference model, we see that the most important and most frequently used information for each respective peer should reside on the peer itself. This will assure easily available information, and allow individual experience to be saved. The best solution to a given problem may be dependent of the users intentions.

Information that does not point to personal references or frequent use, could then be stored in universally available data sources like the Web. Environmental bound information, like activity and context dependent information, could be stored in the environment where it is most likely to be used.

One problem with this scenario is synchronisation of data, which seems like an impossible task when dealing with world wide, continually changing information. A solution to this problem could be the use of *Agents* (Section 5.2.3) that searches the information we need in different information spaces, and returns with the best matching result.

4.2.3 Personalising information

Most of the applications we use, is generally stored on our PC. As *Web 2.0* emerges, we see that more and more applications and hence functionality, moves from the desktop over to the web interface. *Web 2.0* loosely describes a category of websites that are known for interactivity, collaboration and community. This is possible thanks to developments in underlying web technology. One thing that characterises the sites is simplicity. Often, a site is an online application which does one thing extremely well. *Web 2.0* does not just offer remote functionality, it contains sites that offers personalised and specific services. The usual way to tailor a service to an individual user is to save user information and user behaviour in dedicated profiles.

A profile can be a knowledge base on individual information such as service usage patterns, spatially movement patterns, and identity including occupation and interests. With the capabilities of the *Semantic Web* (Section 5.3.1), services could reason over personal information in an autonomous fashion.

A profile would get richer with use, hence provide better service and at the same time allow for thin clients and less user interference. Instead of having all possible services running locally, only a few basic services would be needed to serve as the communication layer towards the environment. All functionality provided by the environment could then be accessed remotely, using the profile as reference for how the service should be used. A user could in addition have different profiles based on, e.g., location and situation.

A profile like this could obviously be vulnerable to abuse. However, much information of this kind is already exposed on the Internet through a range of different *social networking* services. In a way, this is not just a question of privacy, but also about our attitude regarding giving away personal information, which is probably dependent on the generation and habits of the user.

4.3 Cooperation

One of the great envisioned benefits of pervasive computing is the ability of in situ cooperation between heterogeneous digital equipment . To be able to cooperate in such a way, there are some basic properties that must be present.

4.3.1 Sharing resources

One of the ways to support a cooperative environment is to make existing resources available for everyone to use. This means that every peer should share available resources such as processing power and possible services residing on peers. One example could be a location service. This is the kind of resource that every peer probably would posses. If we need to know the coordinates of another peer, we could just ask this peer's *Location service*. In this way, we use another peer's resources to gain access to information. This kind of access should of course be restricted according to personal privacy settings, along with, e.g., how much bandwidth and processing power we are willing to share with others. Peer-to-peer resource sharing is also dependent on people's will to serve the greater good, just like in traditional distributed peer-to-peer networking.

Besides using resources residing on mobile peers, resources like location bound sensors and actuators should in principle be equally accessible to anyone with the ability to discover and use them.

4.3.2 Solving conflicts

If a resource gets queries from more peers than it can handle, we have a conflict. How to solve conflicts depends on many different factors. Humans usually have the ability to negotiate back and forth based on how they prioritise different characteristics, until a reasonable solution is achieved. We want to obtain this property through some kind of autonomous negotiating service.

One approach to this, is the *Arbitration service* described in our depth study [75] and implemented in Part III. As a foundation to the prioritising process, we listed in our depth study some characteristics that seems reasonable to compare against each other.

4.3.3 Cooperative planning

In situ, cooperative planning and execution of workflows is a very complex area within ubiquitous computing. Volatile connections and different interests amongst parties are just as penetrating as in human sense. For peers to work together on a task, they need to agree on a common goal and delegate the activities in a reasonable fashion regarding available resources and will to participate. For this to be possible, we need to have a workflow (Section 5.5.4), or similar construct, with the necessary activities, constraints, and success criteria defined.

The *Scheduling service* (Chapter 15) plan how workflows can be combined to achieve one or several goals. After the planning process, it schedules run-time with specific resources needed to execute each activity.

CHAPTER 5

Supporting technologies and terms

The movement from the computer desktop to the ubiquitous paradigm, as described by Weiser [99], implies that the computer system should now adapt to the user's situation, instead of the user adapting to the computer [56].

Situation adaption, or services and products customisation/personalisation, consists of two major components. Contextual information is required along with some mechanism to reason about this information. When these two components are present, a context-aware system will be able to deliver context-sensitive services to the users [56].

In this chapter we present context and technologies that will help in gathering contextual information and reason about it. In addition we present techniques for automatic situation assessment through learning, mediation in conflicts, construction of dynamic work plans, and scheduling of resource use.

5.1 Context and context-awareness

The use of context is a very important part in ubiquitous computing. Applications that wish to support this paradigm have to be *context-aware* or *context-adaptive* in order to respond to their environment. Or as [56] explains: when interaction can occur anywhere at any time it is imperative that the system adapts to the user in whatever situation the user is in. Before we start the implementation of such an application, we must therefore gain a better understanding of what context is and how it can be used.

Definition of context used in the literature falls into two main categories: those who define context by specific entities, such as location or object; and those who look at context from a more conceptual viewpoint, and focuses on the relationships and structure of contextual information (the formal approach) [56].

The term context-aware was first introduced by Schilit and Theimer [82]. They refer to context as *location*, *identities of nearby people and objects*, and *changes to those objects*. This places their definition in the first category. Several definitions simply provide synonyms for context; for example, referring to context as the environment or situation. These ways of defining context are difficult to apply [33]. An overview of some of these definitions can, however, be found in [25].

The more formal definitions of context include Schmidt et al. [83], which defined context as:

> *Knowledge about the user's and IT device's state.*

The well known definition from Dey and Abowd [34] stated that:

> *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 the application themselves.*

Both definitions described focus on the idea that context is some particular type of information.

In the latest definition, the term context is used in the broadest possible sense; it encompasses any information that might be useful for defining the user's situation [55]. There are potentially many more types of contextual information available than what is used to define a given situation. Hence, a situation is described by a context, which is an instance of the contextual information available. Kofod-Petersen and Aamodt [55] stores such information in a CBR case (Section 5.4.1) so it can be used for case-based situation assessment. This relates to how we envision using CBR to perceive information (Section 4.1.3).

Greenberg [46] argues that context is a dynamic construct and that Dey and Abowd's definition therefore only works for simple and highly routine contextual situations.

Based on the domain independent nature of the *AmbieSense* system (Section 6.9), Kofod-Petersen and Mikalsen [56] gives an extension of Dey and Abowd's context definition:

> *Context is the set of suitable environmental states and settings concerning a user, which are relevant for a situation sensitive application in the process of adapting the services and information offered to the user.*

Brézillon and Pomerol take the view that there is no special type of knowledge that can objectively be called context, they argue that context is in the eye of the beholder, hence, particular kinds of knowledge can be considered context in one setting and domain knowledge in another [20, 56].

To facilitate the flexibility these context definitions give, [54, 55] presents an open context model, first described by [44] which defines the taxonomic structure used in the design phase of the *AmbieSense* system (Figure 5.1).

In [34], Dey and Abowd also defines *primary* context types for characterizing the situation of a particular entity. These are *location*, *identity*, *activity*, and *time*. These can further be used as indices to find *secondary* context about an entity.

Context-awareness is a term used for devices that have access to information about the circumstances under which they operate and can react accordingly. Context-aware devices

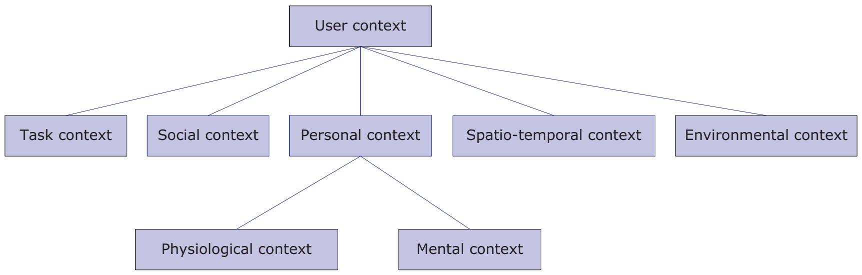


Figure 5.1: Context in the *AmbieSense* project (based on [54, 55, 56])

may try to make assumptions about the user's current situation and use this to act on the users behalf, relieving her from cognitive and trivial tasks.

There have been several attempts on defining context-aware systems. Most of them are too specific to be used in a generic way. More general, and widely referenced, is the definition provided by Dey [33]:

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

Dey and Abowd [34] uses the results of previous work [81, 71] to generalise and categorise the features of context-aware applications. The result is three categories of features that a context-aware application can support:

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

Henricksen et al. [50] has created an infrastructure that facilitates a variety of common tasks related to context-awareness such as modelling and management of context information. They present a model of context for ubiquitous computing that is able to capture features such as diversity, quality, and complex relationships amongst context information.

Many more context-aware applications, frameworks, and middleware have been developed. Some of the most interesting contributions regarding our own implementation is covered in Chapter 6. Several other approaches are surveyed in [54, 9, 58].

5.2 Cooperation in ubiquitous computing

For peers to be able to cooperate, information needs to be exchanged. In a pervasive environment we want this communication to be dynamic and to support heterogeneity.

In this section, we describe some possible techniques for cooperation and interchanging of information.

5.2.1 Peer-to-peer communication and collaboration

At the heart of realising and implementing services for a ubiquitous environment lies the concept of peer-to-peer communication. This network structure supports in its nature the creation of volatile connections with surrounding resources (peers), and releases peers from a centralised, less dynamic architecture.

Today's small mobile devices have an enormous potential of computing power when co-operating, which can and should be utilised for a shared purpose using a peer-to-peer approach [65]. Compared to a stationary computer, a PDA has very limited resources. Letting mobile devices work peer-to-peer and avoiding hot spots (which could easily become a bottleneck) would increase the resource capacity significantly.

However, mobile peer-to-peer networks, such as *Mobile ad-hoc networking* (MANet)[1], face obstacles related to their volatile nature. While desktop computers working in peer-to-peer mode are still relatively stable bound to an environment, mobile devices will encounter and loose other devices frequently. Applications therefore have to tolerate the sudden coming and going of individual peers.

Working peer-to-peer also means a different and more dynamic type of cooperation compared to working in a hierarchical relationship. Devices have to find each other, and have to realise what functions the other peers offer.

Avoiding a hierarchical and centralised structure leads to several benefits in mobile peer-to-peer networks [65]:

- Peer-to-peer systems as distributed systems are more reliable. This is because multiple peers exist for one service.
- They also avoid single network paths that might become congested or unavailable.
- Peer-to-peer systems have better network performance and require smaller bandwidths. Bottlenecks can be avoided by appropriate routing algorithms between peers, for instance in a cluster architecture.
- Administrative and configuration efforts for centralised infrastructures can be avoided in a self-configuring system.
- Shared resources can be used more efficiently. If a peer does not need a special resource it could offer it to other peers. In exchange, a peer can get needed resources from other peers.
- Peers with the same interface can communicate directly without an additional environment.

These benefits are important in ubiquitous computing, as they get the most out of resource-poor devices, and allow for creation of volatile, in situ networks.

To reduce the complexity of many connections, limited network range can help to reduce directly coupled peers. This can also be used to limit the boundaries of a *Smart space* (Section 3.3).

[1] A MANet can be defined as "Network of nodes that just happen to be near each other having no fixed infrastructure" [108]

5.2.2 CSCW

Computer supported cooperative work (CSCW) is a synergy of computer science and a range of theoretical and applied human sciences which studies the field of every day computer use [19]. It is a way to digitally help in the coordination of work activities, mostly by synchronisation. Cooperation and coordination can happen both in asynchronous or in synchronous manner as well as distributed in time and space.

Cooperation support in software is not an easy thing. The complexity is made clear by the fact that activities may be distributed both in time and space, with a varying number of participants, complex workfields to support, and different specialisations of the participants.

The most challenging and complex part is the definition of tasks. In addition, task dependencies drive the need to coordinate work activities [23]. As discussed in Section 5.5, tasks change while working with them. Work processes can be planned thoroughly in advance, but when put into action they will most likely be performed ad-hoc. This ever changing tree of tasks and how they are performed is very complex to implement in CSCW.

A technique to compute coordination requirements is described by Cataldo et al. [23]. In the process, they also revealed that coordination requirements are highly volatile, and frequently extended beyond team boundaries.

To ensure the best possible support in CSCW systems, some guidelines have been elicited regarding management of task interdependencies and common information spaces:

- A way to incorporate task interdependencies is to facilitate mutual awareness among users. When a user change the state of a work process, this must be conveyed to others. If tasks in a work process are tightly coupled and interdependent, tasks may be adjusted according to changes already made in other tasks in the process, possibly by other participants. This approach is necessary in situations where cooperation need to be instant. In workflows (Section 5.5.4) with loosely coupled tasks, it is not equally important with instant awareness of changes made by other participants. In this case, changes will propagate over time.
- It should also be possible to monitor other users, e.g., users working in the same virtual room or in the same work process. This is especially useful in situations where entities are cooperating or competing.
- The restructuring of plans is most likely performed regularly and it should support communication between users, means of getting the others attention, and a way to negotiate re-planning.
- Users need to be able to share and locate information made available by others.
- Information from old work processes should be made available for reuse.
- Domain dependent use of terms should be avoided in general, but not removed as it is also an important part of cooperation.

CSCW systems are often characterised by the means of a two-times-two matrix, separating synchronous and asynchronous communication, and distributed users and non-distributed users (Figure 5.2).

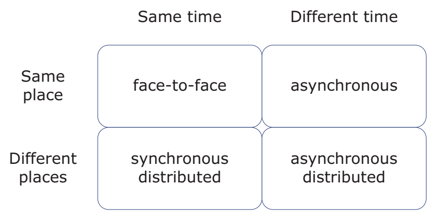


Figure 5.2: A matrix characterizing CSCW by time and space. Inspired by Johansen [52].

Some systems, like e-mail and revision control systems support *asynchronous distribution* because they allow people to collaborate being at different places and dispersed in time. Instant messaging belongs in *synchronous distributed* because it lets people communicate in different places but real-time, while localized systems let people collaborate in the same place but in different time.

The ideal of a full functional CSCW would be to support all of the cells in the matrix.

5.2.3 Agents

Even though there are no definitive agreement of what an agent is [38], a common comprehension is that software agents are programs that assist people and act on their behalf [60]. Agents let people delegate work to them, and can be divided into two groups based on how they function. *Stationary agents* executes only on the system where they are initialised. If they need information which is not on that system or need to interact with agents on a different system, they typically use a communication mechanism such as *Remote procedure call* (RPC) or variants over this (Figure 5.3). *Mobile agents*, on the other hand, have the unique ability to transport themselves from one host in a network to another (Figure 5.4). This ability allows mobile agents to move to a system that contains services they will interact with, and then take advantage of being in the same host or network as the service [60].

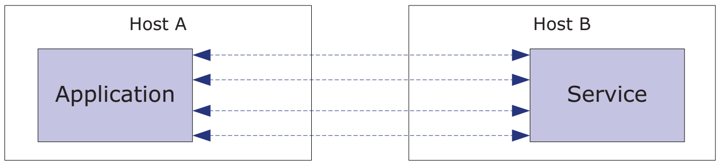


Figure 5.3: RPC-based approach (based on [60]).

No one is required either to deliver information to the agent or to consume any of its output. The agent simply acts continuously in pursuit of its own goals. Agents can also have the ability to communicate with other agents, travel from one host to another (mobile), learn and act according to previous experience. In contrast to the software agents of object-oriented programming, agents are active entities that initiates the communication rather than the contrary.

The following reasons for using mobile agents are mentioned in [60]:

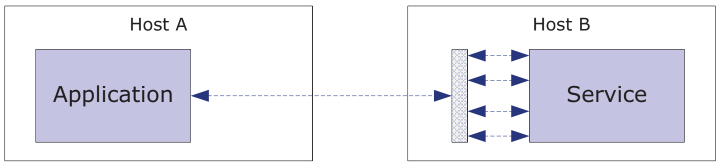


Figure 5.4: Mobile agent-based approach (based on [60]).

- Mobile agents reduce network load, because they operate locally on the remote host instead of communicating back and forth over the network.
- They provide an effective means of overcoming network latency. This is because a mobile agent resides where the task is performed, and therefore reacts faster to changes in its runtime environment. Mobile agents bring us closer to real time behaviour.
- Encapsulation of protocols: If we need to use a special proprietary protocol, this can be established by mobile agents as a "channel" between the hosts without the hosts needing to support this protocol natively.
- They execute asynchronously and autonomously. By embedding tasks into mobile agents and let them execute independently in the network, we do not need an open connection which may be expensive and fragile. Instead, we reconnect with the agent to collect it, e.g. after the task is complete.
- Mobile agents adopt dynamically. Because they can sense their execution environment, mobile agents can react autonomously to changes and distribute themselves to where the configuration is optimal for solving a particular problem.
- They are naturally heterogeneous, because they are generally computer independent and transport-layer independent.
- They are robust and fault tolerant. Mobile agents react dynamically to unfavourable situations and events. If a host is being shut down, agents on that host will be given time to emerge to another host in the network.

These benefits reveal that mobile agents are well suited when delegating activities in a workflow to several peers in the network, or when using cyber foraging. Protocol encapsulation opens for using proprietary protocols for transmission of information. When using mobile networks where the business model is *pay per transmitted amount of data*, mobile agents could be a significant cost-saver by reducing network traffic in comparison to RPC-based approaches. Their ability to react to events in a host and emerge to other hosts is a very useful quality. E.g., in the case of a location-bound task executing in a volatile network where peers connect and disconnect, the task could continually be performed at the peers with the most suitable coordinates.

Multiagent systems

Service-oriented computing (SOC) uses ideas from, and are therefore deeply connected with, *multiagent systems* (MAS). SOC brings considerations such as the necessity of modelling autonomous and heterogeneous components in uncertain and dynamic environments.

Such components must be autonomously reactive and proactive, yet able to interact flexibly with other components and environments. As a result, they are best thought of as agents which collectively form MASs [51].

The key MAS concepts are reflected directly in those of SOC:

- Ontologies (simplified representations of knowledge in a domain, developed with the purpose of facilitating interoperation) (Section 5.3.2).
- Process models (simplified representations of activities and their enactment) (Section 5.5).
- Choreography (simplified business protocols through which services can interact).
- Directories and facilitators (simplified "middle agents" from MASs).
- Service-level agreements and quality-of-service measures (automated negotiation and flexible service execution in dynamic environments).

According to [38], the term "multiagent system" is applied to a system comprising the following elements:

- An environment
- A set of situated, passive objects which can be manipulated by the agents.
- An assembly of agents (specific objects) representing the active entities of the system.
- An assembly of relations, which link objects (including agents) to each other.
- An assembly of operations, making it possible for the agents to perceive, produce, consume, transform, and manipulate objects.
- Operators with the task of representing the application of the operations and how the context changes due to this.

Rational agents have a belief-desire-intention (BDI) model which recognizes the primacy of beliefs, desires, and intentions in rational action. BDI architectures have their root in philosophical tradition of understanding practical reasoning, the process of deciding, moment by moment, the action to perform to facilitate the goal [101]. The BDI model have three distinct strengths [109]:

- An underlying philosophy based on practical reasoning in humans.
- A software architecture that is implementable in real systems.
- A family of logics that support a formal theory of rational agents.

A BDI logic called LORA (Logic of rational agents) contains a temporal component which allows representation of the dynamics of how agents and their environment changes over time [109]. It also includes a component which allows for the representation of agents actions and their effect. This reflects somehow *reinforcement learning*, which are described in Section 5.4.5.

The most interesting application areas for MAS, regarding the ubiquitous domain, are those regarding pervasive service environments. In these, services are widely available everywhere and can be developed independently. Services are dynamically selected, engaged, composed, and executed in a *context-sensitive* manner [51].

5.3 Semantics

Every device, program, or service that wants to interconnect and exchange information, needs to find and understand each other. Most solutions attack the problem at a structural or syntactic level and rely heavily on standardisation of a predetermined set of functionality descriptions. *JINI*, which connects services, and *Universal Plug and Play*, which connects devices, are standards based on this approach. Drawbacks with standards like these are that they are not very dynamic, and get outdated as new functionality needs comes along. This is possible to prevent by instead basing the approach on semantics, i.e., base the communication on common meaning.

5.3.1 The Semantic web

The Semantic Web provides a common framework that spans several interesting techniques and solutions for allowing data to be shared and reused on a global scale. It is developed with the Internet in mind, but on the conceptual level it seems suitable for the ubiquitous domain and contains interesting concepts for our implementation. We explore the components of the Semantic Web, with the original *Scientific American* article [16] as a starting point.

Most of the Web's content today is designed with the intent of being read by humans, not for computer programs to use and manipulate meaningfully. The Semantic Web is an extension of the current Web that will bring structure to the meaningful content of Web pages, creating an environment where software agents (Section 5.2.3), roaming from page to page, can readily carry out sophisticated tasks for users.

The Semantic Web aims to transform the Web from a medium of documents for people, to a medium for data and information that can be processed automatically, better enabling computers and people to work in cooperation.

For the Semantic Web to function and fulfil its purpose, the following must be satisfied:

- Knowledge must be represented as structured collections of information together with sets of inference rules that can be used to conduct automated reasoning.
- A language must be provided that expresses both data and rules, and that allows rules from any existing knowledge-representation system to be exported onto the Web.
- The logic of the inference rules must be powerful enough to describe complex properties of objects but not so powerful that agents can be tricked by being asked to consider a paradox.

Two important technologies that the Semantic Web is based upon are *eXtensible Markup Language* (XML) and the *Resource Description Framework* (RDF). XML allows users to add an arbitrary structure to their documents but says nothing about what the structures mean. Meaning is expressed by RDF, which encodes it in sets of triples, each triple being like the subject, verb, and object of an elementary sentence. These triples can be written using XML tags.

In RDF, a document makes assertions that particular things have properties with certain values. Subject and object are each identified by a *Universal Resource Identifier* (URI)[2], just as used in a link on a Web page. The verbs are also identified by URIs, which enables anyone to define a new concept, a new verb, just by defining a URI for it somewhere on the Web.

The triples of RDF form webs of information about related things. Because RDF uses URIs to encode this information in a document, the URIs ensure that concepts are not just words in a document but are tied to a unique definition that everyone can find on the Web.

Different terms may be used for a common meaning. Ideally, we must have a way to discover such common meanings in the data material. A solution to this problem is provided by the third basic component of the Semantic Web, collections of information called *ontologies* (Section 5.3.2). In this context an ontology is a formal definition of the relationship among terms. The most typical kind of ontology for the Web has a *taxonomy* and a set of inference rules. These inference rules provide deductions based on associations in the dataset, and in that way provide results that are useful and meaningful to the human user. The use of ontologies makes it much easier to develop programs that can tackle complicated questions whose answers do not reside on a single Web page.

To avoid that we are provided with false information, we must be able to check for proof of the deductions performed. A way of doing this is to let a service translate its internal reasoning into the Semantic Web's unifying language, and then let an inference engine in our computer verify the result. For manual control, one could show the relevant Web pages.

As a means to exploit information gathering from several Web pages and to exchange proofs written in the Semantic Web's unifying language, agents (Section 5.2.3) are used. Another vital feature are digital signatures. Agents should be sceptical of assertions that they read on the Semantic Web until they have checked the sources of information.

Many automated Web-based services already exist without semantics, but other programs such as agents have no way to locate one that will perform a specific function. This process, called service discovery, can only happen when there is a common language to describe a service in a way that lets other agents "understand" both the function offered and how to take advantage of it. The Semantic Web, in contrast, is more flexible. The consumer and producer agents can reach a shared understanding by exchanging ontologies, which provide the vocabulary needed for discussion. Agents can even "bootstrap" new reasoning capabilities when they discover new ontologies. Semantics also makes it easier to take advantage of a service that only partially matches a request.

[2]URIs have global scope. Associating a URI with a resource means that anyone can link to it, refer to it, or retrieve a representation of it [86]. In addition, URIs can point to anything, including physical entities, which means we can use the RDF language to describe devices such as cell phones and TVs. Such devices can advertise their functionality, what they can do, and how they are controlled, much like software agents. (URLs, Uniform Resource Locators, are the most common type of URI.)

In a typical course of events, subassemblies of information are passed from one agent to another, each one ”adding value,” to construct the final product requested by the end user. Agents can exploit AI technologies in addition to the Semantic Web. But the Semantic Web provides the foundation and the framework to make such technologies more feasible.

In addition to describing the meaning of content on the web, semantic descriptions of device capabilities and functionality will let us achieve adaption and automation with minimal human intervention also in a ubiquitous environment.

The first concrete steps have already been taken in this area, with work on developing a standard for describing functional capabilities of devices (such as screen sizes) and user preferences. Built on RDF, this standard is called Composite Capability/Preference Profile (CC/PP). Initially, it will let cell phones and other nonstandard Web clients describe their characteristics so that Web content can be tailored for them on the fly. Later, when we add the full versatility of languages for handling ontologies and logic, devices could automatically seek out and employ services and other devices for added information or functionality.

5.3.2 Ontologies

There exists several definitions of an ontology, many contradicting each other [68]. According to [16], ontologies are collections of information. More specific, it is stated that artificial-intelligence and Web researchers have co-opted the term for their own jargon, and that for them, an ontology is a data model that formally represents a set of terms within a domain and the relations between these. In [67], ontologies are defined as common conceptualisations of knowledge. In [86], it is claimed that ontologies are attempts to more carefully define parts of the data world and to allow interactions between data held in different formats.

According to [68], an ontology defines a common vocabulary for researchers who need to share information in a domain. It includes machine-interpretable definitions of basic concepts in the domain and relationships among them.

Many disciplines have developed standardised ontologies that domain experts can use to share and annotate information in their fields. The most typical kind of ontology has a taxonomy and a set of inference rules.

From a more technical view, an ontology is a formal explicit *description of concepts in a domain* (classes), *properties of each concept* (slots) describing various features and attributes of the concept, and *restrictions on slots* (facets) [68]. An ontology together with a set of individual instances of classes constitutes a knowledge base.

Classes are the focus of most ontologies. A class can have subclasses that represent concepts that are more specific than the superclass. Slots also describe properties of instances.

In addition to RDF, WC3 provides a *Web Ontology Language* OWL, developed to use with the *Semantic Web* (Section 5.3.1). OWL checks an ontology to see whether it is logically consistent or to determine whether a particular concept falls within the ontology. OWL uses the linking provided by RDF to allow ontologies to be distributed across systems [86].

Developing an ontology

There are several reasons to develop an ontology, as stated in [68]:

- To share common understanding of the structure of information among people or software agents. This is one of the more common goals in developing ontologies.
- To enable reuse of domain knowledge. This is one of the driving forces behind the development of ontologies.
- To make domain assumptions explicit. It will be possible to change these assumptions easily if our knowledge about the domain changes.
- To separate domain knowledge from the operational knowledge. By doing this, ontologies can function as easily exchangeable algorithm inputs.
- To analyse domain knowledge. Formal analysis of terms (declarative specifications) are extremely valuable when both attempting to reuse existing ontologies and extending them.

Often an ontology of the domain is not a goal in itself. Developing an ontology is akin to defining a set of data and their structure for other programs to use. **Problem-solving methods**, **domain-independent applications**, and software agents use ontologies and knowledge bases built from ontologies as data.

Ontology development is an iterative process. A stepwise suggestion on how to create an initial ontology is presented in [68]:

Step 1. Determine the domain and scope of the ontology.

Step 2. Consider reusing existing ontologies.

Step 3. Enumerate important terms in the ontology.

Step 4. Define the classes and the class hierarchy.

Step 5. Define the properties of classes – slots.

Step 6. Define the facets of the slots.

Step 7. Create instances.

After these steps are complete, we will almost certainly need a revision of the initial ontology. This is then followed by an iterative process that will most likely continue through the entire lifecycle of the ontology.

5.4 Learning and reasoning

Artificial Intelligence (AI) is concerned with intelligent behaviour in artifacts. Intelligent behaviour, in turn, involves **perception**, **reasoning**, **machine learning**, **communicating**, and **acting in complex environments** [67]. In this section, we look at some AI techniques which appear as reasonable candidates to our proof-of-concept implementation.

5.4.1 Case-based reasoning

Case-based reasoning (CBR) is a body of concepts and techniques that touch upon some of the most basic issues related to knowledge representation, reasoning, and learning from experience [87]. Reasoning is usually modelled as a process where conclusions are drawn by chaining together generalised rules, starting from scratch. In CBR, the primary knowledge source is not generalised rules but a memory of stored *cases* recording specific prior episodes [62]. CBR exploits analogies and similarities with these previously solved cases to solve new ones [67].

The roots of case-based reasoning in AI dates back to the early 1980s and is found in the works of Roger Schank on dynamic memory and the central role that a reminding of earlier situations and situation patterns has in problem solving and learning [3, 78]. In the same era, Janet Kolodner pioneered CBR, emphasising its use in situations of real-world complexity [63].

At the Norwegian University of Science and Technology (NTNU), Agnar Aamodt and colleagues at Sintef have studied the learning aspect of CBR in the context of knowledge acquisition in general, and knowledge maintenance in particular. We use mainly the work of Aamodt and Plaza [3] to describe the field more thoroughly in the following.

Concept of CBR

A case-based reasoner solves new problems by adapting solutions to older problems. Generally a case-based reasoner will be presented with a problem and it searches its memory of past cases (case base) and attempts to find a case that has the same problem specification as the case under analysis. If the reasoner cannot find an identical case in its case base, it will attempt to find a case or multiple cases that most closely match the current case.

In situations where a previous identical case is retrieved, assuming that its solution was successful, it can be offered as a solution to the current problem. In the more likely situation that the case retrieved is not identical to the current case, an adaptation phase occurs. During adaptation, differences between the current and retrieved cases are first identified and then the solution associated with the case retrieved is modified, taking these differences into account. The solution returned in response to the current problem specification may then be tried in the appropriate domain setting.

At the highest level of abstraction, a case-based reasoning system can be viewed as a black box that incorporates the reasoning mechanism and the following external facets (Figure 5.5).:

- The input specification or problem case
- The output that defines a suggested solution to the problem
- The memory of past cases, the case base, that are referenced by the reasoning mechanism

In most CBR systems, the *case-based reasoning mechanism* has an internal structure divided into two major parts: the case retriever and the case reasoner (Figure 5.6). The case retriever's task is to find the appropriate cases in the case base, while the case reasoner uses the cases retrieved to find a solution to the problem description given. This reasoning process generally involves both determining the differences between the cases

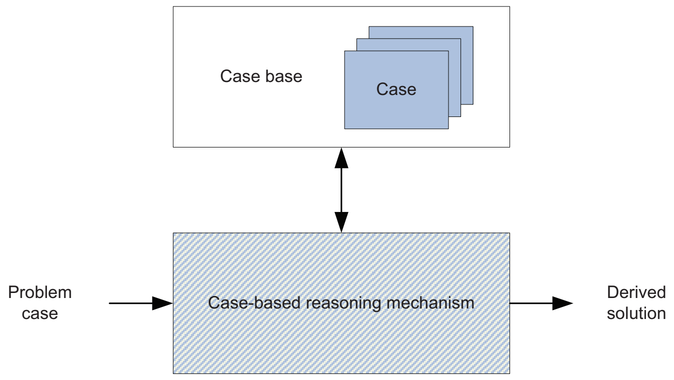


Figure 5.5: CBR system (based on [87]).

retrieved and the current case, and modifying the solution to reflect these differences appropriately. The reasoning process may or may not involve retrieving additional cases or portions of cases from the case base.

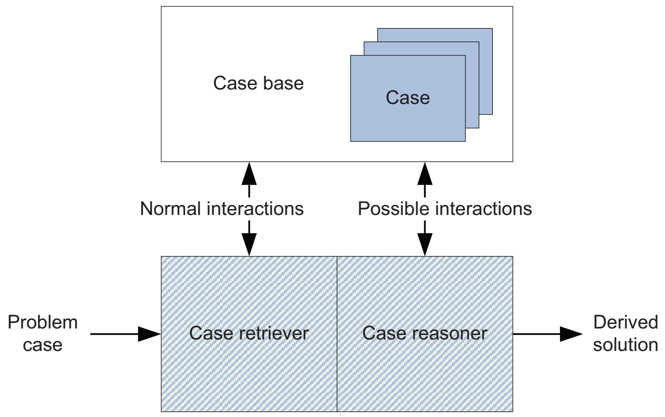


Figure 5.6: Two major components of a CBR system (based on [87]).

Case-based reasoning has been formalized for purposes of computer reasoning as a four-step process [3]:

1. **Retrieve:** Given a target problem, retrieve cases from memory that is relevant for solving it. A case consists of a problem, its solution, and, typically, annotations about how the solution was derived.

2. **Reuse:** Map the solution from the previous case to the target problem. This may involve adapting the solution as needed to fit the new situation.

3. **Revise:** Having mapped the previous solution to the target situation, test the new solution in the real world (or a simulation) and, if necessary, revise.

4. **Retain:** After the solution has been successfully adapted to the target problem, store the resulting experience as a new case in memory.

These steps are part of the CBR life cycle (Figure 5.7), which represents the *process-oriented view* of the descriptive framework presented by Aamodt and Plaza [3]. The process is supported by supplying the cases with general knowledge (which is usually domain dependent).

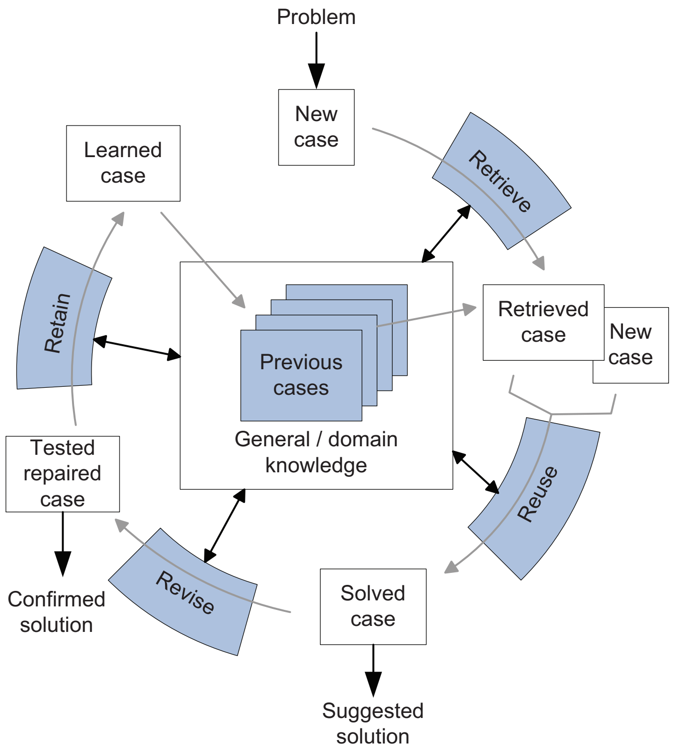


Figure 5.7: Process oriented view of the CBR life cycle (adopted from [3]).

While the *process-oriented view* provides a global and external view of the CBR process, the *task-oriented view* decompose and describe the four top-level steps, where each step is viewed as a task that the CBR reasoner has to achieve (Figure 5.8). In the figure, tasks are named in bold letters, while methods are written in italics. The links between task nodes appears as plain lines and indicates task decompositions. The top-level task is problem solving and learning from experience and the method to accomplish this task is case-based reasoning (indicated in a special way by the stippled rectangle). The top-level task is split into the four major CBR tasks corresponding to the four processes of Figure 5.7, retrieve, reuse, revise, and retain. All the four tasks are necessary in order to perform the top-level task.

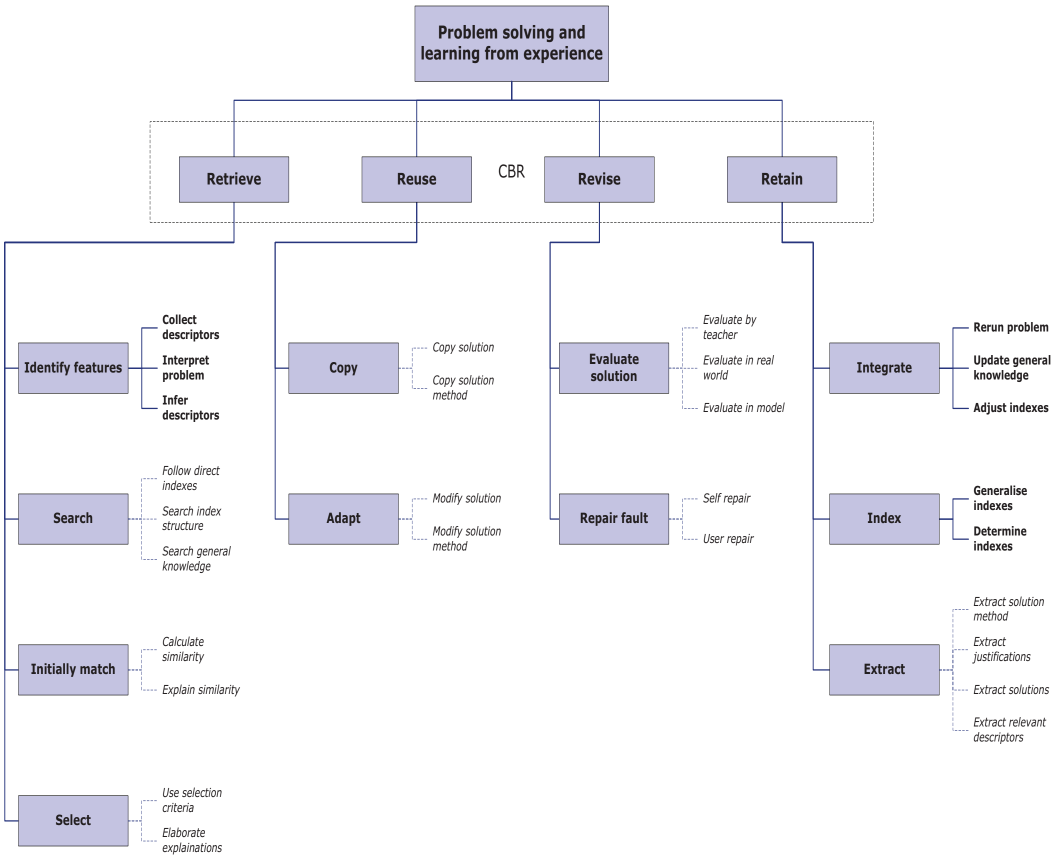


Figure 5.8: Task-method decomposition of CBR (adopted from [3]).

CBR methods

Actually, "case-based reasoning" is just one of a set of terms used to refer to a range of different methods for organizing, retrieving, utilizing and indexing the knowledge retained in past cases. Cases can be stored and indexed in different ways and structures. How a case is matched and applied to the present problem also varies. CBR methods may be purely self-contained and automatic, or they may interact heavily with the user for support and guidance of its choices. Some CBR methods assume a rather large amount of widely distributed cases in its case base, while others are based on a more limited set of typical ones. Past cases may be retrieved and evaluated sequentially or in parallel [3].

The main types of CBR methods mentioned in [3] are:

Exemplar-based reasoning:
: In this method, a concept is defined extensionally, as the set of its exemplars[3]. Solving a problem is a classification task, i.e. finding the right class for the unclassified exemplar. The class of the most similar past case becomes the solution to the classification problem. Modification of a solution found is outside the scope of this method.

Instance-based reasoning:
: A specialization of exemplar-based reasoning into a highly syntactic CBR-approach. It requires a relatively large number of instances in order to close in on a concept definition, because it lacks general background knowledge. A major focus in this method is to study automated learning, free from user intervention. Because of this, the representation of the instances are usually simple (e.g. feature vectors). Basically, this is a non-generalization approach to the concept learning problem addressed by classical, inductive machine learning methods.

Memory-based reasoning:
: Emphasizes a collection of cases as a large memory, and reasoning is a process of accessing and searching in this memory. The access and storage methods may rely on purely syntactic criteria, or they may attempt to utilize general domain knowledge.

Case-based reasoning:
: A typical case is usually assumed to have a certain degree of richness of information contained in it, and a certain complexity with respect to its internal organization, i.e., not a feature vector holding some values and a corresponding class. Typical case-based methods are also able to modify, or adapt, a retrieved solution when applied in a different problem solving context, and they utilises general background knowledge. Core methods of typical CBR systems borrow a lot from cognitive psychology theories.

Analogy-based reasoning:
: Sometimes used as a synonym to case-based reasoning. However, it is also often used to characterise methods that solve new problems based on past cases from a different domain, while typical case-based methods focus on indexing and matching strategies for single-domain cases. The major focus has been on reuse of past cases, which is called the mapping problem: Finding a way to transfer, or map, the solution of an identified analogue to the present problem.

[3]An exemplar is a well known usage of a scientific theory. It is the well known solution to some puzzle of the research field, models of excellence.

Comparison with human reasoning

Schank states in [79] that taking case-based reasoning seriously, as a cognitive model, implies that experience plays a fundamental role in human learning as well. When humans encounter a problem, they often try to solve it by using a human equivalent of CBR, or vice versa. Facing a new problem, a person will most likely refer to a past experience from a similar problem. Learning from experiences is the fundamental process of case-based reasoning. People can also learn from other people's experience through oral or written account [79]. The similarities may cover the entire case or only the points that led to a portion of the result. Another part of the case may oppose to the compared case. This is referred to as interpretive reasoning.

Comparison with rule-based systems

Rule-based systems (Section 5.4.2) has a rule base consisting of a set of production rules of the form: **IF** A **then** B where A is a condition and B is an action. In addition, such systems have an inference engine that compares the data it holds in the working memory with the condition parts of rules to determine which rules to fire.

One of the most time consuming tasks when developing a rule-based system, is the knowledge acquisition task. The need for collecting and converting domain-specific information into some formal representation can sometimes be a huge task, especially if the domain is not well understood.

Case-based systems usually require significantly less knowledge acquisition. This is because they involve collecting a set of previous experiences without the added necessity of extracting a formal domain model from the cases. Another benefit of CBR is that a system can be created with a small or limited amount of experience, and then be created incrementally as cases become available.

Challenges

As Aamodt and Plaza points out in [3], there are no universal CBR methods suitable for every domain of application. The challenge is to come up with methods that are suited for problem solving and learning in particular subject domains and for particular application environments. Core problems addressed by CBR research concerns the area of *knowledge representation* and methods for *retrieval*, *reuse*, *revision*, and *retainment*.

Examples of how these problem areas have and can be dealt with are given in, amongst others, [3] and [57]. Pal and Shiu does it by introducing us to *Soft case-based reasoning* (SCBR) which we elaborate in the following.

Soft Case-based reasoning

There has been much progress since the early days of CBR. This includes the understanding of concepts such as similarity, relevance, and materiality. In [87], Pal and Shiu investigates if it is possible to achieve a quantum leap in capabilities of CBR systems through traditional computing and reasoning methods. As an answer and alternative to this, they bring us the notion of Soft Case-Based Reasoning (SCBR) which is based on soft computing, a computing methodology that is a coalition of methodologies which collectively provide a foundation for the conception, design, and the utilization of intelligent systems. The principal members of the coalition are *fuzzy logic*, *neurocomputing*, *evolutionary computing*, *probabilistic computing*, *chaotic computing*, *rough set theory*, *self-organizing maps*, and *machine learning*.

By combining and using different methods from these areas, Shiu and Pal propose possible solutions to the challenges mentioned, except they use some different terms. Instead of relating to the *reuse* and *revise* stages (which in practical application are difficult to distinguish), they replace and combine these into an *adaption* stage. The term *retain* is replaced by *case learning* and *case-base maintenance*. We give an overview of the methods used in traditional CBR and those used in SCBR, based on [87] (Table 5.1).

Table 5.1: Traditional methods and SCBR methods for addressing the challenging areas of CBR.

	Traditional methods	**Soft computing techniques**
Case representation	• relational • object-oriented • predicate	• fuzzy sets • rough sets
Case selection and retrieval	clustering and classification techniques: • weighted feature-based similarity • weighted Euclidean distance • Hamming and Levenstein distances • cosine coefficient • k-nearest neighbour principle	fuzzy clustering and classification techniques: • weighted intracluster and intracluster similarity • fuzzy ID3 classification • fuzzy c-means clustering • feature weighting – gradient descent and neural networks – genetic algorithms • neural networks • neuro-fuzzy model • rough self-organising maps

(continued from previous page)	**Traditional methods**	**Soft computing techniques**
Case adaption	strategies: • reinstantiation • substitution – constraint-based – feedback-based • transformation methods: • learning adaption knowledge from case data • rule- and case-based reasoning for adaption • adaption matrix • configuration techniques	machine learning: • fuzzy decision tree • back-propagation neural network • Bayesian model • support vector machine • genetic algorithms
Case-base maintenance	• qualitative maintenance • quantitative maintenance	• rough-fuzzy approach • fuzzy integral approach

Criticism

Critics of CBR argue that it is an approach that accepts *anecdotal evidence*[4] as its main operating principle. Without statistically relevant data for backing and implicit generalization, there is no guarantee that the generalization is correct [105].

5.4.2 Rule-based expert systems

Rule-based expert systems are knowledge based systems which can implement actions for given situations. They have a set of definable rules with criteria which match up with knowledge stored in the systems. Each time knowledge is updated in the system, an

[4]Anecdotal evidence is argumentation based on data that is too scarce for statistical relevance.

inference engine[5] will try to se if any of the defined rules match the thumb-print of the knowledge represented in the system. If a match is found, the corresponding action will be performed. Actions may induce new knowledge in the system which, in turn, may match other rules. This goes on until no more rules can be matched successfully.

Knowledge in the system is represented by facts. Facts, or knowledge, are distinctive, concrete values, which can be interpreted by the inference engine. Facts can become known to the system by direct or procedural input. See figure 5.9 for the basic structure of an expert system.

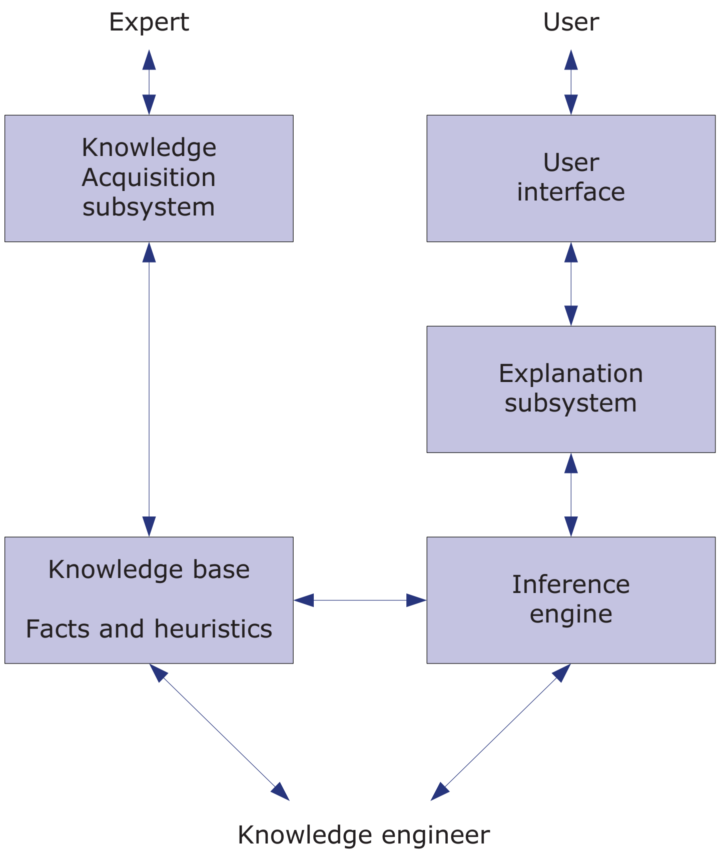


Figure 5.9: Basic structure of an expert system. Borrowed from [67].

Humans and computers are fairly distinct in the way we solve complex problems. The motive for making an expert system is therefore to try to imitate the way humans make their reasoning.

This would also greatly help people without knowledge of programming to make new rules in the system, which helps the system to reason. An example of such a rule could

[5]An inference engine match given rules to the facts, known as knowledge, and reports matches. The inference engine consists of all the processes that manipulate the knowledge base to deduce information requested by the user [67].

be formulated as:

> ***If*** *temperature sensor* ***and*** *heating oven* ***and*** *temperature lower than 20℃* ***then*** *turn on heating oven*

and

> ***If*** *temperature sensor* ***and*** *heating oven* ***and*** *temperature higher than 21℃* ***then*** *turn off heating oven*

Simple rules like these are something everyone can express and would be fairly similar to how we think.

Rules like these are possible to implement in conventional programming languages but requires considerable effort because it demands a high level of dynamics, pattern matching, and insurance of a minimal computational amount. Several expert systems have therefore been developed to support this kind of operations.

The terms *temperature sensor*, *heating oven*, and *temperature* have to be known to the system by means of programming it. The same applies to the actual process of turning on and off the heating oven. The use of this functionality, on the other hand, can be programmed by rules that are easily understandable to anyone and that resembles how we think.

A rule-based system is able to learn and make more and more complex reasoning based on the adding of new, easily definable rules to the system.

5.4.3 Bayesian networks

A Bayesian network (or a belief network) is a probabilistic graphical model that represents a set of variables and their probabilistic dependencies in a concise and typically tractable way [104]. It is very similar to *expert systems* (Section 5.4.2), but instead of working with true and false logic, like if-then rules, it works with probability theory.

It will then be made up by rules of the following form:

> ***If*** *A is true,* ***then the probability*** *that B is true is X.*

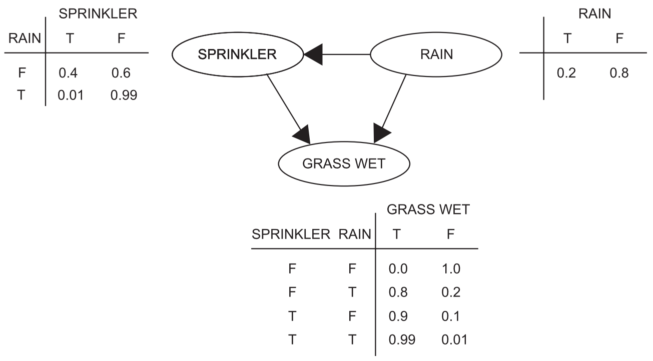
Instead of saying; "if it is cloudy outside, then it will rain", you can say; "if it is cloudy outside, then the probability of rain is 85%".

In general, Bayesian networks are used for reasoning with uncertain information. An alternative for this kind of reasoning, among others, is *fuzzy logic* (Section 5.4.4).

Formally, a *Bayesian network* is a directed, acyclic graph (DAG) whose nodes are labelled by random variables [67]. Nodes can represent any kind of variable, be it a measured parameter, a latent variable or a hypothesis. Efficient algorithms exist to perform inference and learning in Bayesian networks [104]. Bayesian networks are sometimes called *causal networks* because the arcs connecting the nodes can be thought of as representing direct causal relationships. To construct a Bayesian network for a given set of variables, we draw arcs from cause variables to immediate effect.

A simple Bayesian network is shown in figure 5.10. Here, the set of variables consists of three variables; *rain*, *sprinkler*, and *grass wet*. The arcs show how the variables affect each other, and the tables show their probabilistic dependencies. All variables has two possible values; *true* and *false*.

We see that rain affects the use of the sprinkler, and that both these variables affect the wetness of the grass. Because we in this case know the probabilistic dependencies between cause and effect, the model can answer questions like "What is the likelihood that it is raining, given the grass is wet?", however, we do not provide the formula here (see [104] for details).



	SPRINKLER	
RAIN	T	F
F	0.4	0.6
T	0.01	0.99

RAIN	
T	F
0.2	0.8

		GRASS WET	
SPRINKLER	RAIN	T	F
F	F	0.0	1.0
F	T	0.8	0.2
T	F	0.9	0.1
T	T	0.99	0.01

Figure 5.10: Bayesian network example (borrowed from [104]).

Advantages of Bayesian networks is that they are memory saving, and it is intuitively easier for a human to understand direct dependencies and local distributions than complete joint distribution.

Bayesian networks are used for, among other areas, decision support systems.

5.4.4 Fuzzy logic

The request for further work in [66] claims that it would be possible to use fuzzy logic (FL) in the inference engines of a context-aware workflow system to handle a wider array of context states without involving an exception handler. This is because inference engines originally only deal with absolute values of true or false, while fuzzy logic provides the possibility of partial truths.

We will look at fuzzy logic as a mean for not being restricted to use crisp values where this is not beneficial. The following section will give an introduction to fuzzy logic and provide an example on how to use it.

History and range of use

In 1965 Lotfi Zadeh published a paper called "Fuzzy Sets" and later stated the principle of incapability:

> *As the complexity of a system increases, our ability to make precise and significant statements about its behaviour diminishes until a threshold is reached*

beyond which precision and significance become almost mutually exclusive characteristics.

In recent years, fuzzy logic and fuzzy control have been used in many real world applications and is now an established subject in the areas of control, decision, and knowledge engineering [11].

Why should we need to use fuzzy concepts? We interpret the world in terms of classifications expressed in linguistic forms invented by humans. We use fuzzy concepts to avoid sharp boundaries between similar cases. Some trees are definitely trees and not bushes. Others are less definitely trees and more possibly bushes [11].

FL provides a simple way to arrive at a definite conclusion based upon vague, ambiguous, imprecise, noisy, or missing input information. FL's approach to control problems mimics how a person would make decisions, only much faster [96].

Various shades of grey are allowed in addition to black and white. Truth can vary in intensity. There can be varying degrees of temperature, consistence, and so forth. These can be clustered into respectively linguistic terms such as hot, cold, hard, soft, and so on. These groupings are clusters introduced to simplify our understanding of the real world and to encourage meaningful communication between people. All the clusters represent fuzzy concepts. Remove the fuzziness and you approximate. This approximation can be very dangerous because our decision foundation may be impaired, [11].

Fuzzy logic provides flexibility in knowledge representation and a robustness in the methods of inference. An important part of the inferential robustness is the ability of fuzzy logic to generalize from prototypical cases to new situations. This process of induction is extremely important in human intelligence. Fuzzy sets can be used in constructing a theory of generalization, a theory of **case-based reasoning**. This theory will be a method for interpolating from similar but known situations [11].

Fuzzy sets

A fuzzy set is a range of values that in varying degrees belongs to a term. To clarify this, we have produced the following example, which will follow you through and help to explain FL.

Water can be cold, comfortable or hot. Comfortable can be anything from, say, 25℃ to 35℃ while cold and hot is respectively below and above. You can even have degrees of it, like slightly, very and extremely cold or hot. There may also be that the water feels slightly cold and slightly comfortable. How can we represent this with numbers?

The answer is overlapping sets. These sets can be seen as functions that define the degree of belonging to a specified term, where the x-axis represents the value range and the y-axis represents the degree of belonging to the specified term in a range of 0 to 1.

Let's return to the water example and try to illustrate it. Examine Figure 5.11. Here the term *comfortable* is defined as 25℃ to 35℃, *cold* as anything below 20℃ and *hot* anything above 40℃.

The sets of the tree terms overlap. This means that something can actually be a little bit of both. If we take the temperature 22℃ it is slightly *comfortable*, but also slightly *cold*.

Now, the only thing a computer can reason about this is that, if we ask it how does 39℃ feels, it would say that it is mostly hot but has a little degree of comfort to it as well.

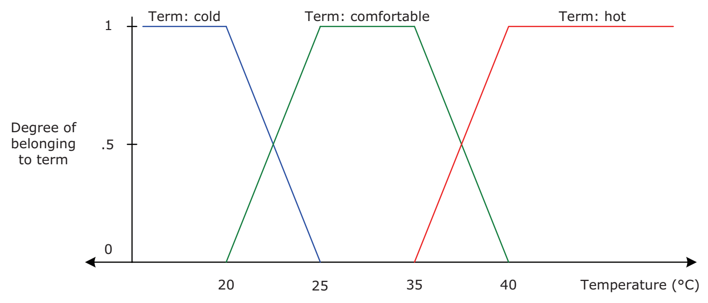


Figure 5.11: Fuzzy sets of the water temperature example.

This resembles how we human think and reason. This example is a very simple one and more complex resolution can be done, but for introduction we will stick to the simple one.

To complete the example we will look at a method to adjust the resulting temperature of a water flow based on ${}^{litres}/_{minute}$ of hot water mixed with a constant flow of cold water. To do this we need to define fuzzy sets describing the change of flow of hot water as well.

Change in hot water flow will be described by the terms *none* defined as $0^{litres}/_{minute}$, *decrease* as reducing the flow with $1^{litres}/_{minute}$ to $10^{litres}/_{minute}$ and *increase* to raise the water flow with $1^{litres}/_{minute}$ to $10^{litres}/_{minute}$. An illustration of this can be seen in Figure 5.12

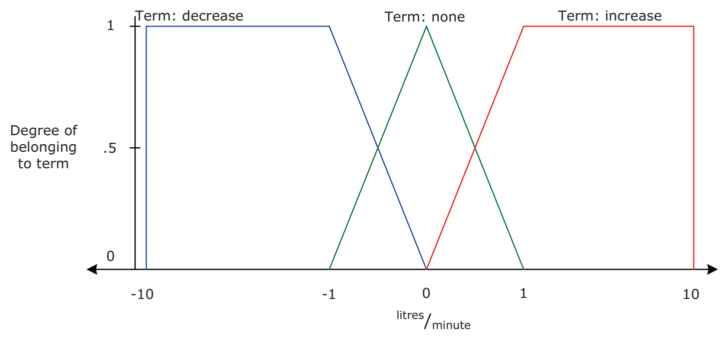


Figure 5.12: Fuzzy sets of a flow of water.

Applying rules

Now we have the base for writing rules. The whole point of fuzzy logic is to mimic the way we think. Rules are therefore easy to formulate using normal English.

We have already stated that the flow of cold water is constant. The temperature of the hot and cold water is also constant. The resulting temperature of the mixed water is therefore reliant on the flow of the hot water. Three rules have to be defined to control the process of regulating the temperature.

1. **If** water temperature **is** *hot* **then** change of hot water flow **is** *decrease*.
2. **If** water temperature **is** *cold* **then** change of hot water flow **is** *increase*.
3. **If** water temperature **is** *comfortable* **then** change of hot water flow **is** *none*.

These rules do what they say. If the temperature is cold, increase the hot water flow, if it is hot, decrease the hot water flow and if it is comfortable, change nothing.

This is easy to implement with crisp values and rules as well, but it would not give smooth transitions. With crisp values the temperature would, from the smallest change in value, suddenly pass from being comfortable to cold. Fuzzy logics have smooth transitions all the way.

Getting a crisp result - Defuzzification

The command to the actuator performing the change of hot water flow has to be a crisp value. How do we get that from the fuzzy sets and application of rules? If the transitions is to be smooth, it has to be able to change the water flow fast if its very cold and more slowly if its only slightly cold.

The fuzzy example we have been looking at has one input value, temperature, and one regulating value, hot water flow. It is fully possible to have more than one input value, it is actually normal, but for simplicity we will stick to one.

To achieve a crisp output value, whenever an input value changes, it is tested against the defined rules. For each rule it matches a result there will be produced and combined to one crisp value.

A rule matches if the input value belongs to the fuzzy set. E.g., if the input temperature is 22℃ it matches both rule 2 and 3. The crisp result will then be a combination of change of water flow terms *decrease* and *none*.

Let us look at the match of rule 2, illustrated in Figure 5.13. If we draw a line from 22℃ and upwards, we find that it intersects the graph for *cold* at 0.6 on the y-axis. This is how much the temperature matches the term and also how much of the answer for rule 2 we should incorporate in our answer. We therefore draw a line and fill everything underneath 0.6 of the term *high* for hot water flow.

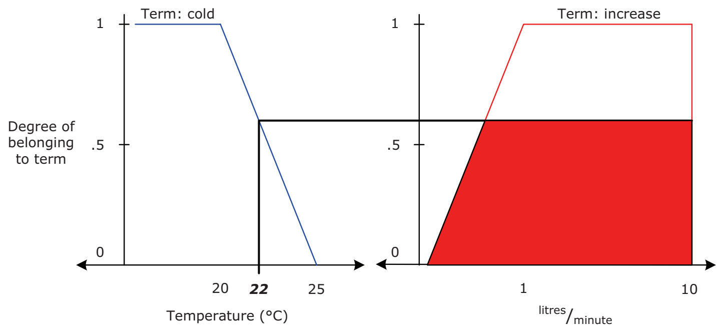


Figure 5.13: Result of match of rule 2.

We also do the same for rule 3, since it matches that as well. Here we only have 0.4 match and mark everything underneath the result for rule 3, the term *none*. This is illustrated in Figure 5.14.

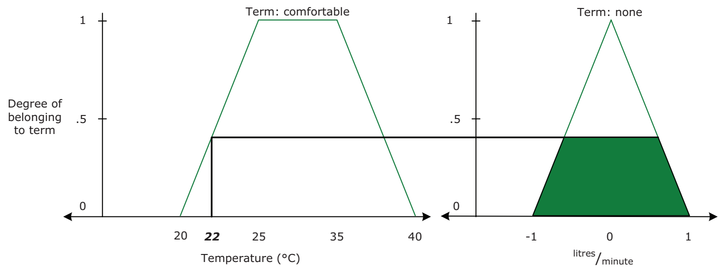


Figure 5.14: Result of match of rule 3.

No more rules match the input value of 22℃ and we therefore can conclude by joining the two results together (see Figure 5.15).

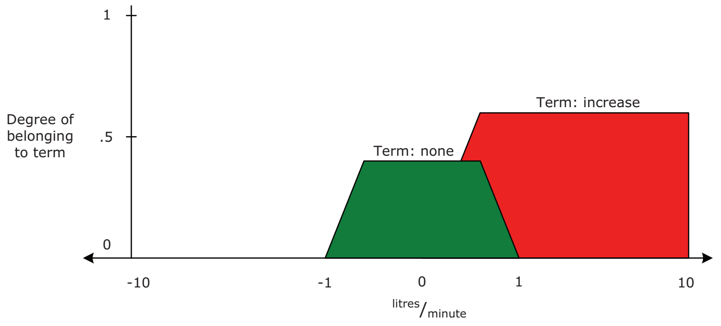


Figure 5.15: Combination of results of match for rule 2 and 3.

The normal way of calculating the result is by superimposing the results upon each other, forming a single geometric shape and then calculating the centroid. The x-axis value of the centroid is then the crisp value. In our case the result would be ≈ 5. This means that if the temperature is 22℃ it would increase the hot water flow with $5^{litres}/_{minute}$. A new temperature would then be read and the process starts again until the temperature is only in the term *comfortable*, only rule 3 fires, and the water flow is held constant.

5.4.5 Reinforcement learning

Learning from interaction is a fundamental idea underlying nearly all theories of learning and intelligence. In [93] we are given an introduction to *reinforcement learning*, which is a computational approach to learning whereby an agent tries to maximise the total amount of reward it receives, when interacting with a complex, uncertain environment.

The agent learns what to do by mapping situations to actions, and study the result of these actions. The learner is not told which actions to take, as in most forms of *machine learning*, but instead must discover which actions yields the most reward by trying them. Actions may affect not only the immediate reward but also the next situation and, through that, all subsequent rewards. *Trial-and-error* and *delayed reward* are the two most important distinguishing features of reinforcement learning.

Interaction with the environment provides us with information about cause and effect, consequences of actions, and what to do in order to achieve goals. Such interactions are undoubtedly a major source of knowledge about the environment and ourselves. Reinforcement learning is much more focused on goal-directed learning than other approaches to *machine learning*.

Reinforcement learning is not defined by characterising learning methods, but by characterising a learning problem. The basic idea of the reinforcement learning problem is simply to capture the most important aspects of the real problem facing a learning agent, interacting with its environment to achieve a goal. Such an agent must be able to sense the state of its environment to some extent, and must be able to take actions that affect the state. The agent must also have one or several goals relating to the state of the environment. The formulation is intended to only include just these three aspects: sensation, action, and goal.

As opposed to *supervised learning*, which is learning by provided examples, an agent using reinforcement learning is able to learn from own experience. This introduces a tradeoff between exploration and exploitation. The agent has to exploit what it already knows, but it also have to explore in order to make better action selections in the future. In practice this is a progressive approach where the agent favour the alternatives that seems best.

Another key feature of reinforcement learning is that it explicitly considers the whole problem of a goal-directed agent interacting with an uncertain environment. This is in contrast with many approaches that consider subproblems without addressing how they fit into the big picture.

5.4.6 Artificial neural networks

Artificial neural networks, also known as parallel distributed processing networks, are interesting mainly for their ability to learn [67]. However, they do not learn from own experience, but rather from examples provided to them through *supervised learning* [93].

More formally, a neural network is a computing solution that is loosely modelled after cortical structures of the brain. It consists of interconnected processing elements called nodes or neurons that work together to produce an output function. The output of a neural network relies on the cooperation of the individual neurons within the network to operate. Processing of information by neural networks is characteristically done in parallel rather than sequentially as in earlier binary computers or Von Neumann machines.

A unique property of a neural network is that it can perform its overall function even if some of the neurons are not functioning. This is because it relies on its member neurons collectively to perform its function. In other words, it is robust regarding tolerance of error or failure. Additionally, neural networks are more readily adaptable to *fuzzy logic* (Section 5.4.4) computing tasks than Von Neumann machines [103]. A conceptual description of a neural network is given in figure 5.16.

Neural networks are trainable systems that can learn to solve complex problems from a set

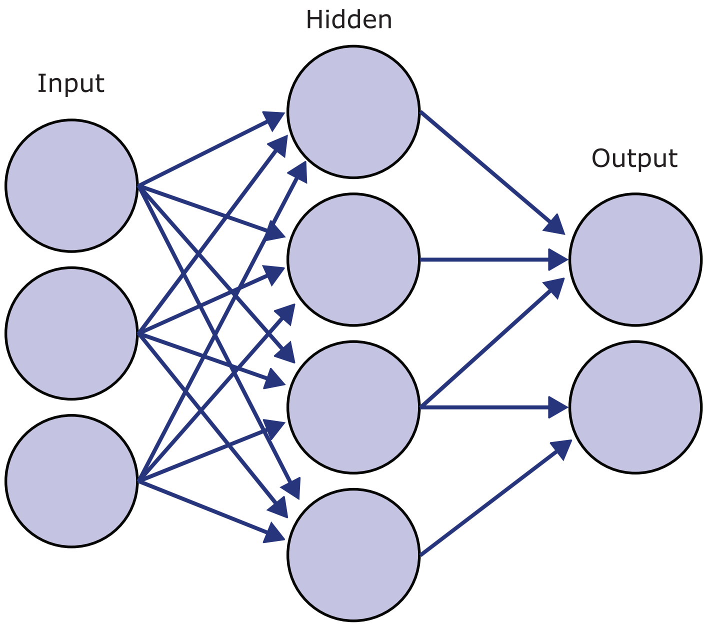


Figure 5.16: Conceptual view of an artificial neural network (borrowed from [103]).

of provided examples, and generalize the acquired knowledge to solve unforeseen problems. I.e., they are *self-adaptive systems*. A neural network does not have to be adaptive, but its practical use comes with algorithms designed to alter the strength (weight) of the connections in the network to produce a desired signal flow.

While traditional AI uses computational algorithms to solve problems, neural networks use *networks of agents* (Section 5.2.3) as the computational architecture to solve problems.

When it comes to the efficiency of today's neural networks, current research in theoretical neuroscience are targeting the question of how complex, and what properties that individual neuro elements should have in order to resembling animal intelligence. Recently, scientists from IBM created an artificial neural network which contained the simulated equivalent of the number of neurons in an actual mouse cortex ($8 * 10^6$), but with less synapses (6300 instead of 8000 per neuron). The project was run on a BlueGene/L supercomputer, but still each neuron was only able to fire about ten times slower than in real life and the project contained only approximations of one type of neurons [43]. The shortcomings of this cutting edge project compared to real life implies that we do not have the knowledge or the technology available yet, to simulate life, but results are promising for a distant future.

One obstacle for imitating real life is that the brain is massively parallel, even more so than advanced multiprocessor computers. This means that simulating the behaviour of a brain on traditional computer hardware is necessarily slow and inefficient. Neural networks are based on efforts to model information processing in biological systems, which may rely largely on parallel processing as well as implicit instructions based on recognition of patterns of *sensory* input from external sources. Rather than sequential processing and execution, at their very heart, neural networks are complex statistic processors.

Even though we have not come so far in simulating real life using artificial neural networks, these nets have a great utility through the ability to infer a function from observations. The tasks to which artificial neural networks are applied to, tend to fall within the following broad categories:

- Function approximation, or regression analysis, including time series prediction and modelling.
- Classification, including pattern and sequence recognition, novelty detection and sequential decision making.
- Data processing, including filtering, clustering, blind signal separation and compression.

5.5 Process modelling

Process modelling for computers have been looked upon as a sequence of tasks to be performed in ordered succession. This way of ordering work processes has been criticised because it does not take into account the informal practice that often or always occurs during execution of a plan [12]. This means that the plan is not followed and is not of much help or even guidance, thus making the workflow system incapable of actually supporting the users.

When we perform work, we often do it in an *ad hoc* fashion. This means that we do not follow strict plans for how to do things, but instead adapt to the given situation. On the other hand, we have plans describing almost any major activity we perform. This is called the planning paradox [13].

We will in this section investigate some of the different theories underlying process modelling, to form a basis for our suggested planning and scheduling architecture, the *Scheduling service*.

5.5.1 Activity theory

Several approaches to examine activity have been proposed. Some examples of this are *Actor-network theory* [61], *Situated actions* [92], and the *Locales framework* [40]. According to Kofod-Petersen and Cassens [22], the works of Vygotsky and Leont'ev proposes the use of activity theory to model context and to describe situations, which is an interesting theory.

To understand the *planning paradox*, we must look at one of the core concepts of activity theory, *human activity*. Human activity has three basic characteristics [12], which are:

1. Directed towards a material or ideal object. The object distinguishes one activity from another.
2. The interaction with the object is done by mediating tools, language, etc.
3. How activities are performed varies from one culture to another.

Human knowledge about the world is a result of our interaction with objects. *Human activity* can be described as a hierarchy of *activities*, which consists of one or more *actions*. Actions in turn consists of one or more *operations*, as illustrated in Figure 5.17.

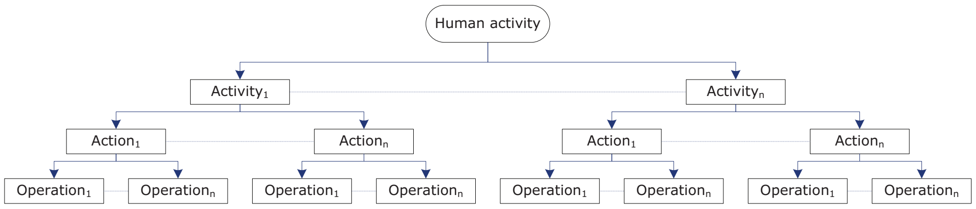


Figure 5.17: The descriptive hierarchy of human activity.

As a result of how human characterize activities, an activity performed on an object is done to satisfy a need. The human reflections on the activity and the expected result are the *motive* of the activity. Actions are controlled by conscious *goals* which are the human's anticipation of the action's result. Actions are often *poly-motivated*. This means that two or more actions can temporarily merge, motivating the same action. Each *operation* is effectuated by the concrete physical *condition* of the action.

All three levels of human activities are guided by motivation. The anticipated result guides the activity and the perception of the environmental state of the activity. Memory of prior results forms a person's anticipation. If the measure of anticipation and result yields differences, it gives rise to new knowledge and experience.

This hierarchical description helps to understand the fundamental role planning has in human cognition and activities. Based on prior experience, we plan our actions to realise the activity. These plans are implemented through operations which are adjusted to the concrete physical *conditions* of the actions (the context state). This adjustment makes the operations situated or contextually influenced.

It is important to note that this hierarchical composition is not fixed over time. If an action fails, the operations comprising the action can get conceptualised, they become conscious operations and might become actions in the next attempt to reach the overall goal. This is referred to as a *breakdown situation* [54] (Section 4.1.3). In the same manner, actions can become automated when done many times and thus become operations. In this way, we can model a change over time.

An expanded model of activity theory, *Cultural Historical Activity Theory* (CHAT) (Figure 5.18), covers the facets of social and cultural context. This model expands the basic triangle of mediation, which shows how an activity is composed of a subject, an object, and a mediating artifact or tool. A subject is a person or group engaged in an activity. An object is held by the subject and motivates the activity. Since we consider social activities, the acting subject is part of a community. The relations between the subject and the community, as well as between the community and the object are mediated by a set of rules and the division of labour [54]. Rules are accumulations of knowledge about how to something.

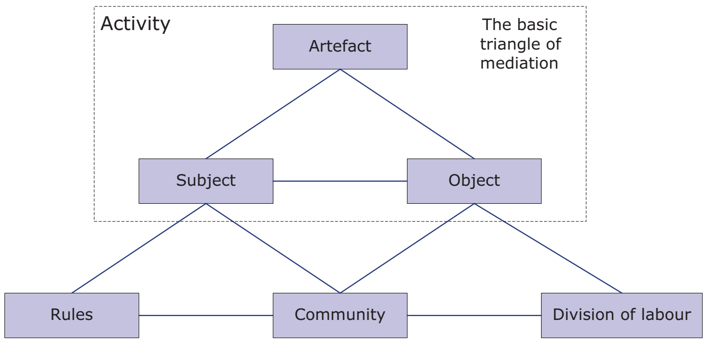


Figure 5.18: Cultural Historical Activity Theory (based on [54])

5.5.2 Situated planning and actions

The process of decomposing goals and workflows into activities, builds on several assumptions about future results. Typically, problems arise when these assumptions are not right. These deviations from anticipated results are however not exceptions, but normal, and form the basis for learning and enhancing future actions. Based on this, a plan is only a resource used in the making and execution of an activity, and is subsequently enhanced.

Bardram discuss in [12] how plans are made out of *situated actions*, and in turn are realised *in situ*. He claims that work can be characterised as *situated planning*, an understanding which is backed up by *activity theory* (Section 5.5.1) which emphasise on the connection between plans and the contextual conditions for realising these plans.

Based on activity theory, a plan can be defined as [12]:

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

Some guidelines are also given in [12] about which characteristics a computer tool supporting situated planning should posses:

- Producing and altering plans in the course of work:
  In order for plans to become resources for the future realisation of an activity, the plan should be made as a part of the activity. The tool should support ongoing creation and modifications of plans based on experience.

- Sharing plans within a work practice:
  Plans should be shared between actors involved in an activity to support coordination and cooperation.

- Executing plans according to the conditions of the work:
  Since the anticipated result and actual result often differentiate, the tool should support altering or skipping of actions based on the user's need.

- Inspecting plans and their potential outcome:
  A tool should provide a way for the users to see the potential outcome of the plan

and to see the actions it consists of and their needed resources. This can be done by simply displaying a list or by simulating the execution of the plan and being able to alter it. The plan should also reveal the conditions under which the plan is useful. Our solution is practically built around this idea.

- Monitoring the execution of plans:
  Having an overview of the unfolding of activities is essential. The tool should recognise and support any deviations from the plan. This should also include any initial deviations. The ability to trace backwards to the original plan should also be supported.

Plans represents anticipation of how work will be performed, while actual work itself is *ad-hoc* [91]. A plan therefore becomes a resource as guidance to how to proceed while rarely followed strictly. Plans are generated as a result of work as well. If work has been performed successfully, a plan may be constructed representing the work, acting as a plan the next time around.

Nevertheless, plans are important resources in handling collaboration, coordination, and execution of complex tasks as they ensure that no important parts are forgotten and are thus widely and successfully used.

5.5.3 Planning and scheduling

As a core aspect of human intelligence, planning has been studied since the earliest days of AI and cognitive science [1]. Planning can be thought of as determining all the small tasks that must be carried out to accomplish a goal. Planning also takes into account rules (constraints) which control when certain tasks can or cannot be executed. Scheduling can be thought of as determining whether adequate resources are available to carry out the plan [73].

The work of creating plans and schedules are usually based on general goals. The executing entity (agent) must then design plans on how to achieve these goals. The design work can be divided into the following tasks [73]:

- Receive a request for a plan for the next schedule horizon.
- Obtain goals for the next schedule horizon.
- Break goals into smaller tasks needed to accomplish the goals.
- Gather needed information.
- Determine rules (constraints) which govern when certain tasks can or cannot take place.
- Generate the plan and return it.

In a ubiquitous setting, planning and scheduling will in most situations be triggered by location and other context. Hence, the runtime environment is highly dynamic which results in the need of planning and scheduling as continuously running services.

Distributed, continual planning

In [18], Brenner and Nebel describes an approach to *continual planning* in dynamic (multi agent) environments. They emphasize an integration of planning, execution, and monitoring as a continuous cycle. This approach prepares for extension of the executing agent's knowledge as a part of the plan, and then revising the planning decisions in light of the new knowledge.

Through the years there has been changing perspectives on planning that have led to an interest in *Distributed, continual planning* (DCP). Some of these perspectives ignore *context* (Section 5.1) in the planning process, while others tolerate, exploit, or establish planning and execution context [32].

Planning is a very complex area, and simplifications have been made to make planning more feasible. Several assumptions are made in [49, 18], which belongs to the perspective that do not embody context in the planning process:

- An agent is typically assumed to know everything that is relevant to the planning problem and to know exactly how its available actions can change the world from one state into another.
- The planning agent is assumed to be in control of the world, so that the only changes to the state is the result of the agent's deliberate actions.
- The agent's preferred world states are constant throughout a planning episode. It will not change its mind about what goals to achieve while planning or executing the plan.

These simplifications allow the planning problem to be serialized: A planning agent first formulates a plan and then executes it. It is also assumed that the planning and execution for one episode have no conjunction with the planning and execution done for previous or future episodes.

In the real world, however, context needs to be taken into account, and plans do not always proceed as expected. One approach to handling this kind of uncertainty, is to enumerate the possible states that might arise at execution time and plan for each of them. This may result in a very large conditional plan (universal plan) with possible execution alternatives. Which part of the universal plan is executed depends entirely on the environmental context at execution time [17, 84].

If the knowledge available to the agent is insufficient or suggests an intractably large set of possible states, a better approach is to formulate a nominal plan, monitoring progress, and, if deviations, repair by halting the execution and create a revised plan [6, 36].

In complex environments, changing context can make goals and aspects of the world evolve continuously rather than be fixed throughout a planning episode. In this setting an agent should continually evaluate and revise its plans [29].

Continual planning recognizes that plan revision should be an ongoing process rather than one that is triggered only by failure of current plans. It also adopts the perspective of not planning in too much detail too far into the future because evolving circumstances can render such details obsolete. Continual planning, therefore, tolerates the planning and execution context by maintaining flexibility and opportunism [32].

If an agent has the knowledge to do so, it should not only tolerate the presence of context such as other agents, but also exploit it through cooperation. Highly flexible coordination and communication is important in addressing uncertainties in complex, dynamic domains. Flexibility and reusability can be reached by providing agents with general models of teamwork [94]. This opens for a system of agents to engage in distributed, continual planning. The idea of allowing agents to exploit the larger multiagent context for cooperative planning and execution opens the door to purposely establishing such a context to improve what agents can accomplish [90].

Agents that have formulated abstract plans can analyse potential relationships between their possible plans, commit to particular constraints on how they will realize these plans, and then incorporate these influences in their elaboration decisions in a decentralized way [27].

The problem of constructing plans in a distributed environment has been approached from two different directions [32]:

- One approach has begun with a focus on planning and how it can be extended into a distributed environment, where the process of formulating or executing a plan could involve actions and interactions of a number of participants. This approach allows for parallel execution of plans and is referred to as *cooperative distributed planning* (CDP).
- The other approach has begun with an emphasis on the problem of controlling and coordinating the actions of multiple agents in a shared environment and has adopted planning representations and algorithms as a means to an end. This approach is referred to as *negotiated distributed planning* (NDP). The emphasis is not on cooperatively defining the best group plan, as in CDP, but to convice other agents to accommodate its own preferences.

Reasoning and negotiation techniques are needed that allows agents to exploit the opportunities of coordination, refine their plans based on other agents' evolving plans, and modify plans based on contextual information.

5.5.4 Workflow

Automation of work processes is an important property in ubiquitous computing, and is supported by the WfMC reference model. Previous work on *Smart work processes* has looked at how to incorporate context-awareness into workflow standards. We are interested in using workflows in the planning of *schedules* based on context.

WfMC's workflow definitions, shown in the Figure 5.19, tells us that an activity is a mean to ensure certain **goals** or *what is intended to happen*. Activities are further divided into atomic **tasks**, which refer to *what is actually happening*.

We will use workflow definitions in much the same sense as Sørensen, Wang, and Conradi [89]. They state that an activity can be defined in a process model by providing goals, preconditions (constraints) and postconditions (success criteria), invariants, and the use or production of artefacts and resources. Context or context changes can affect an activity in different ways, making it a context-aware activity.

Kofod-Petersen and Cassens [22] states that one of the most important context parameters available in many situations is the activity performed by an entity present in the

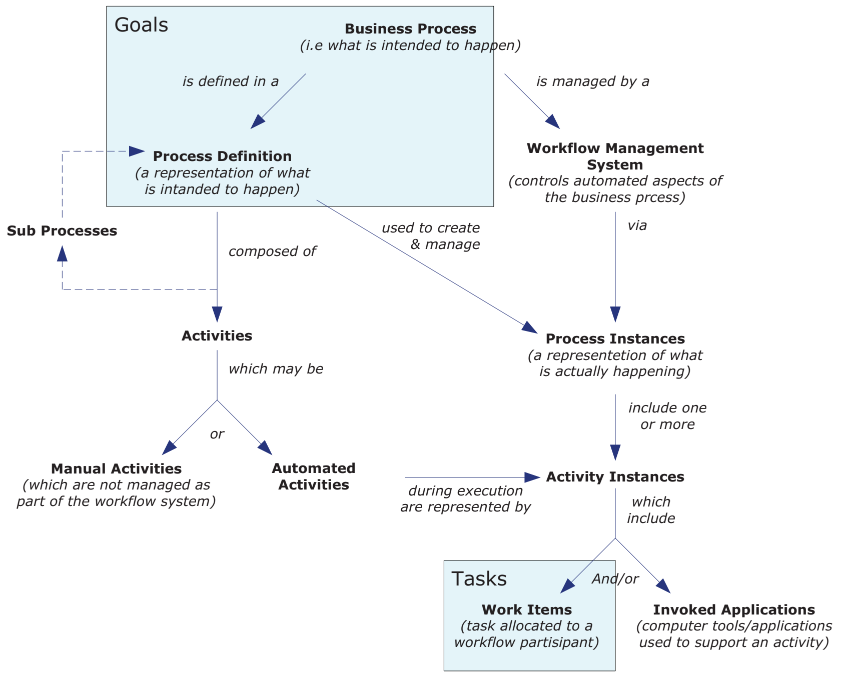


Figure 5.19: Workflow terminology (based on [28])

.

environment. They believe that focusing on activities will gain a better understanding of context and context awareness. This is in accordance with our use of activities, which has a central role in our contribution.

Context-aware workflows are a means to ease the development of context-aware applications. Previous work within the MOWAHS project directed towards *Smart work processes* has looked at how to incorporate context-awareness into workflows. Nødtvedt and Nguyen [66] states that interface 1, 2, and 3 in the WfMC workflow reference model [28] supports being context-aware. Hauso and Røed [48] concludes their thesis by stating that using context in workflows process enactment is feasible and will be increasingly important in the future. Development of prototypes supporting context-aware workflow processes are therefore important research.

Wieland et al. [102] continues this pursuit by suggesting new ways of incorporating context-awareness in workflows. They state that to achieve this, workflow meta-models should allow the modelling of context in workflows and the use of context to control the flow between activities. A workflow managing system should then be coupled to a context-provisioning platform, to become aware of the corresponding process meta-models (representing technical processes).

Our own contribution within the development of context-aware workflows center around the generation of workflows based on contextual state in an automated fashion, and how they adapt to changes in this context state (Chapter 14).

CHAPTER 6

Frameworks and middleware

In this chapter we present the frameworks and middlewares of interest or considered for our proof-of-concept implementation.

6.1 Smart work processes architecture

Mobile work is work processes performed in a mobile environment and dependent of context information extracted from the physical environment [88, 89] (Section 3.2).

Mobile work and context can mutually influence each other; mobile work can change the state of the environment by performing activities, and the environment state are most likely to have an impact on several aspects related to the execution of mobile work.

A way of supporting mobile work is given by Sørensen et al. [88, 89] through the notion of *Smart work processes*.

The concept is a combination of ubiquitous computing and workflow. *Smart work processes* enables adaptation of work processes to dynamic working environments and can be used as a means to coordinate multiple actors. These 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.

This makes *Smart work processes* especially suited for using context information to monitor and coordinate activities within a context-rich environment. Coordination amongst activities needs to be performed between work processes and the work environment. Coordination is also needed between multiple actors performing cooperative or possibly competing work processes, and with respect to some stated paramount requirements like safety, time, and economy.

An activity may be affected by context or context changes in several ways. When activities are dependant on the contextual state, it needs to include specifications and rules for how context relates to the activity, and how context affects the activity. Context states could also trigger adaptation of workflows by changing or introducing new activities. These mechanisms coincide to *situated planning* (Section 5.5.2).

The SWP architecture (Figure 6.1) [88, 89] is a conceptual architecture supporting *Smart work processes*. It consists mainly of components and services that separate concerns related to a sensing, cooperative environment. It also functions as a guide on how to delegate responsibility amongst services, and how they are interconnected.

The SWP architecture will function as a reference model for our work. The architectural solutions we propose can be looked upon as services in the components for cooperation (COWCS) and workflow enactment (CAWES), and these will therefore be emphasized in the following introduction of the different components.

An *actuator* is responsible for changing the environment based on input from an actuator service.

A *sensor* is responsible for measuring and transmitting sensor readings to a context service.

The *actuator service* initiates actuations on the environment by receiving instructions from CAWES and translate these to the correct actuator.

The *context service* provides all communication with sensors, and filters the information it receives to reduce the workload. This is thought done by client-based subscription on context. It is only the implemented services and available sensors that sets the limitations on what kind of context to collect.

The *Client-based, Context-aware Workflow Enactment Service* (CAWES) interacts with the other components directly and indirectly, and is the enactor regarding execution of workflows. It is responsible for monitoring and executing delegated activities or fragments of these. The delegation is taken care of by the *Server-based Workflow Enactment Service* and COWCS.

CAWES is also responsible for trigging new tasks based on surveillance of the present execution state, information from COWCS and the *Server-based Workflow Enactment Service*, and contextual input from the *Context service*.

In addition, CAWES send contextual information about the activity state back to COWCS and the *Server-based Workflow Enactment Service*, and updates the environment by initiating actuations using the *Actuator service*.

CAWES could be extended to also maintain historical data for the purpose of automation of tasks and better utilization of resources. This could be done by saving data from the execution process and use a suited reasoning technology to utilize it at a later stage.

The *Client-based, Cooperative Workflow Coordination Service* (COWCS) is responsible for managing and coordinate activities in a multi-actor environment based on policies. It is the component which other actors interact with, and therefore it should support exchange of process goals, to let actors know if they have convergent activities which may be solved in a cooperative manner. Current state of executing activities should be provided to keep actors up to date and be able to synchronize their tasks.

COWCS has close interaction with CAWES, regarding sharing of resources with other actors. Planned activities should be collectively known because it might influence the ordering of tasks and the use of resources. Coordination needs should be broadcasted because external resources not directly available for the actor might be crucial for the execution of a task. The information it sends to CAWES, is event triggered, based on context reasoning.

The *Server-based Workflow Enactment Service* is also updated by COWCS, and provides a distributed way of communicating with CAWES. Its purpose is to support traditional workflow and high level process management for the involved participants.

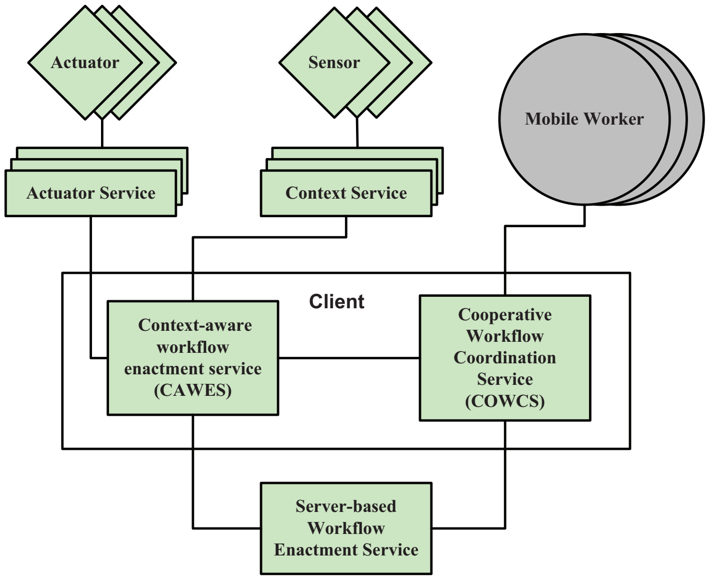

Actuator
Sensor
Mobile Worker
Actuator Service
Context Service
Client
Context-aware workflow enactment service (CAWES)
Cooperative Workflow Coordination Service (COWCS)
Server-based Workflow Enactment Service

Figure 6.1: An architecture for Smart work processes.

6.2 Context Toolkit

The *Context Toolkit* [33, 35] is a context-aware architecture which embrace several functions: It is a design process for context-aware applications, a conceptual framework, and a *Rapid Application Development* (RAD) tool. It aims to add the use of context to existing noncontext-aware applications and to evolve existing context-aware applications.

In the design process you choose which context-aware behaviour to implement, determine what context is required, determine what hardware is required, interpret context, and choose context-aware behaviour.

The conceptual framework consists of the following components:

- BaseObject: communication, global time
- Widget: sensor independent presentation of context
- Service: Widget including manipulation of context
- Discoverer: can be asked for components
- Interpreter: interprets context
- Aggregator: represents summary on several Widgets

The framework is built on the concept of enabling applications to obtain the context they require without them having to worry about how the context was sensed (much like the *SWP* architecture). A *context widget*, an abstraction that implements this concept, is responsible for acquiring a certain type of context information. It then makes that information available to applications in a generic manner, regardless of how it is actually sensed. Applications can access context from widgets using traditional poll and subscribe methods. This functionality is preserved by the *Context service* in the *SWP* architecture.

Context widgets operate independently from the applications that use them. This eases the programming burden by not requiring the application designer to maintain the context widgets, while allowing to easily communicate with them. Because context widgets run independently of applications, there is a need for them to be persistent and available all the time. The *SWP* architecture implements as mentioned a similar tactic, by separating this as a service provided outside of the clients (application) boundaries.

Because an important part of context is historical information, the *Context Toolkit* provides support for the storage of context. *Context widgets* automatically store all of the context they sense, and make this history available to any interested applications. Applications can then use historical information to predict the future actions or intentions of users.

This prediction or interpretation functionality is encapsulated in the *Context interpreter* abstraction. Interpreters accept one or more types of context and produce a single piece of context. An example is converting from a name to an e-mail address. A more complicated example is interpreting context from all the widgets in a conference room to determine that a meeting is occurring.

The *Context Toolkit* makes in other words the distribution of the context architecture transparent to context-aware applications, mediating all communications between applications and components.

Context aggregators aggregate or collect context. It is responsible for all the context for a single entity. Aggregators gather the context about an entity (e.g., a person) from the available context widgets, behaving as a context proxy for applications.

The toolkit's object-oriented API provides a superclass called *BaseObject* which offers generic communication abilities to ease the creation of own components.

6.3 JCAF

The *Java Context Awareness Framework* (JCAF) is a closed source project developed in Java. It is described as a *service infrastructure and programming framework for creating context-aware applications*, developed by Jacob E. Bardram, a professor at the IT University of Copenhagen (http://www.itu.dk/people/bardram/pmwiki/).

The motivation for developing JCAF was the need for more generic programming frameworks supporting context-aware computing, and through this, help programmers develop and deploy context-aware applications faster [14].

The JCAF runtime infrastructure consists of three layers; a context client layer, a context service layer, and a context sensor and actuator layer.

The *Context service* layer consists of several *Context services* which are connected in a peer-to-peer setup, each responsible for handling context in their specific environment. A network of services can then cooperate by querying each other for context information, and it brings the notion of hops when resolving context information, which constricts the propagation over the network.

Context clients are the context-aware applications accessing one or more *Context services*. Clients can add or remove context information, they can add, query for, and use context transformers, and they can adjust the topology of the context service network.

Context sensor and actuators are also implemented. Sensors are components that can update a specific entity in context, where the communication to the governing *context service* is handled. Actuator components can either use the poll strategy or get informed when the desired context information is changed. This way it can perform actuations based on values in context. The entity *heating oven* could have the relations location, surface temperature, and switch. By changing the switch value in context, you could turn on the heating oven.

This infrastructure fits perfectly to the *Smart work processes* architecture [89] (Section 6.1), as they both have the same abstraction of the responsibilities of a *Context service*.

6.4 JXTA

Background

JXTA (Juxtapose) is an open source, peer-to-peer, cross-platform technology created by Sun Microsystems in 2001. It is defined as a set of XML based protocols that allow any device connected to a network to exchange messages and collaborate in spite of the network topology. It was designed to allow a wide range of devices to communicate in a decentralized manner [106]. JXTA can be virtually ported to any computer language as bindings, with the Java binding as the most mature implementation.

Manner of operation

JXTA peers create a virtual overlay network[1] which allows a peer to interact with other peers directly, even when some of the peers are behind firewalls and NATs or use different network transports. In addition, each peer is identified by a unique ID, a 160 bit SHA-1[2] URN[3] in the Java binding, so that a peer can change its localization address while keeping a constant identification number [106]. It is able to create networks across carriers like IrDA and Bluetooth, as well as TCP/IP.

Peers

JXTA defines two main categories of peers: *edge peers* and *super-peers*. The super-peers can be further divided into *rendezvous peers* and *relay peers*.

Edge peers are usually defined as peers that have transient, low bandwidth network connectivity, like mobile entities.

[1] An overlay network is a network built on top of another network.

[2] SHA-1 is a hash function which produces an unique message digest from any given input.

[3] A Uniform Resource Name (URN) is a Uniform Resource Identifier (URI) that uses the urn scheme, and does not imply availability of the identified resource.

Rendezvous peer is a special purpose peer that is in charge of coordinating the peers in the JXTA network. Rendezvous peers are used for instance to forward requestst, discover and cache advertisements, and to bridge between different network segments, even running different transport protocols.

Relay peer allows the peers that are behind firewalls or NAT systems to take part in the JXTA network. This is performed by using a protocol that can traverse the firewall, like HTTP, for example.

It is worth noting that any peer in a JXTA network can be a rendezvous or relay as soon as they have the necessary credentials or network/storage/memory/CPU requirements.

Peer groups

A peer group is a collection of peers that have a common set of interests. A peer group provides a scope for message propagation and a logical clustering of peers. In JXTA, every peer is a member of a default group, NetPeerGroup, but a given peer can be member of many sub-groups at the same time.

Each group should normally have at least one rendezvous peer and it is not possible to send messages between two groups. In our case, only one group is created

Advertisements

Advertisements are programming language neutral metadata structures that describes any resource in a peer-to-peer network (peers, groups, pipes, services, etc). The communication in JXTA can be thought as the exchange of one or more advertisements through the network.

Pipes

A pipe is a virtual unidirectional connection between peers. Pipes are used by JXTA to exchange messages and data. Input and output pipes exist that can be bound at runtime to different peers. A pipe works asynchronously. There are special pipes that can be bound to multiple endpoints.

Architecture

JXTA defines a three layer architecture:

The **JXTA Core** is responsible for basic level operations such as communication. The core includes protocols and provides general security mechanisms. In this way, the JXTA Core is comparable to the kernel of an operating system. It controls features like peer groups (building), peer pipes, and peer monitoring.

JXTA Services support higher-level functions such as searching, file and resource sharing, indexing, and caching. JXTA Services enable features needed for platform independent collaboration.

JXTA Applications use peer services as well as core layer functions. Example applications are content management, shared searching, distributed computing, and instant messaging.

6.5 jCOLIBRI

The Case-Based Reasoning framework jCOLIBRI is a technological evolution of COLIBRI using description logic (DL), and an object-oriented framework in Java. COLIBRI (Cases and Ontology Libraries Integration for Building Reasoning Infrastructures) is a domain independent architecture in *LISP/LOOM*.

jCOLIBRI can use application-independent *ontologies* and *CBROnto*. CBROnto is ontologies that includes task and method knowledge about CBR [31]. Describing methods, it is used to formalize the CBR problem solving methods, that are organized in a library around the tasks they resolve. This allows for *DL* reasoning with the problem solving method descriptions to check their applicability regarding an external context formed by the domain knowledge and the cases.

The design of the framework comprises a hierarchy of Java classes plus a number of XML files organized around the following elements:

- Tasks and Methods. XML files describe the tasks supported by the framework along with the methods for solving those tasks. Customized tasks and methods, based on provided interfaces, can be added.
- Case Base. Different connectors are defined to support several types of case persistency, from file systems to databases. One can easily create own connectors by implementing a given interface and configure an XML file.
- Cases. A number of interfaces and classes are included in the framework to provide an abstract representation of cases that support any type of actual case structure. A case structure consists of all properties a case is built up of, organized in a tree. This definitions is configurable and held in an XML file.
- Problem solving methods. The actual code that supports the methods included in the framework. Building a CBR system is a configuration process where the system developer selects the tasks the system must fulfil and for every task assigns the method that will do the job. Ideally, the system designer would find every task and method needed for the system at hand, so that you would program just the representation of cases. However, in a more realistic situation a number of new methods may be needed and, less probably, some new task. Since jCOLIBRI is designed as an extensible framework, new elements will smoothly integrate with the available infrastructure as long as they follow the framework design. This is done by implementing given interfaces and configuring the XML files.

CommonKADS is the leading methodology to support structured knowledge engineering and is a good candiate for becoming the de facto European standard and point of reference [85]. It regards *knowledge based system* (KBS) development as continuous improvement of a set of *models* [85]. Each is a model of various aspects of the *KBS* and its environment. These sets are separate but collaborating and is reflected in jCOLIBRI as domain knowledge (CBROnto and ontologies) and *problem solving methods* (PSMs) and represent commonly occurring, domain-independent problem-solving strategies [31].

6.6 Creek

Creek is a knowledge intensive case-based reasoning system, or KI-CBR. A traditional CBR case is formed by specific knowledge to solve problems. Creek seeks to combine the specific with general domain knowledge to enhance the description logic reasoning around the cases, the similarity functions and their solutions [2].

The cases are represented as parts of the general domain knowledge network. This renders the system able to reason about the cases, their concepts and their relations, to better calculate case similarities and to gain better knowledge of what the actual case represents. Linking this to the Semantic Web, the system is able to make use of the distributed knowledge from the system. This property is similar to the CBROnto of jCOLIBRI in Section 6.5.

The case-bases, domain models and how theses are related is described with the Creek OWL Vocabulary (http://creek.idi.ntnu.no/owl/).

Being a KI-CBR with a general domain knowledge description of cases, it falls into the same category as jCOLIBRI, but there are some differeces between them [39].

- In jCOLIBRI, the PSMs does not get direct access to the ontologies of CBROnto.
- In jCOLIBRI, there is a separation between domain independent CBR concepts and the knowledge in the domain model used in KI-CBR systems.

6.7 SOCAM

SOCAM proposes an ontology-oriented approach to support context reasoning and context knowledge sharing, and a service-oriented approach to support interoperability between different context-aware systems. It also proposes a formal context model based on ontology using Web Ontology Language (OWL) to address issues including semantic context representation, capturing context classification information, and enabling context reasoning and context knowledge sharing [95].

The middleware uses a central server, called context interpreter, which gains context data through distributed context providers and offers it in mostly processed form to the clients. The context-aware mobile services are located on top of the architecture, thus, they make use of the different levels of context and adapt their behaviour according to the current context [10].

6.8 CoBrA (Context Broker Architecture)

CoBrA is an agent-based architecture for supporting context-aware systems in smart spaces (e.g., intelligent meeting rooms, smart homes, and smart vehicles). It uses the *Web Ontology Language (OWL)* to define ontologies for context representation and modelling. It defines rule-based logical inference for context reasoning and knowledge maintenance, and provides a policy language for users to control the sharing of their private information. Central to *CoBrA* is a server agent called *context broker*. Its role is to maintain a consistent model of context that can be shared by all computing entities in the environment, and to enforce the user-defined policies for privacy protection [26]. A service like this could be implemented in the *context service* component in the *SWP* architecture.

6.9 Ambiesense

AmbieSense is a context-aware system that sets out to give relevant information to the right situation and user [5]. It does this by providing the ambient landscape with intelligent equipment. The system and its reference architecture supports the development of mobile information services that are ubiquitous, personalised and adapted to the situation.

Ambiesense access different information depending of user's interests and the current context and location. To achieve this, the system uses two context technologies, *Context middleware* and *Context tags*. *Context middleware* runs on mobile devices and in the provided *Content service*. Agents personalises and adapts the information system in terms of information extraction, retrieval, filtering, and presentation. Hence, they help the mobile users to get the right information to the right situation.

Context tags is a means of capturing and communicating information about the surroundings and communicate with mobile devices. They automatically send the contextual information about the surroundings to the mobile users who travel. The effect is that the user is relieved from specifying the context around him. Context tags can be networked and integrated with existing computers and wireless network infrastructures.

A user context is capable of describing the user's interests, his state, the social setting, the spatio-temporal aspects, and other entities in the surroundings, like the open context model described by Kofod-Petersen et al. [54, 55, 56] promise (Section 5.1).

The corner-stones of the AmbieSense system are:

- wireless context tags
- mobile devices
- intelligent agents
- personalised and context-sensitive information services

PART III

Arbitration

CHAPTER 7

Arbitration introduction

When two or more actors operate in the same environment, conflicts can occur. Instead of just discovering conflicts ad-hoc, it is preferable to also be able to discover them based on scheduled workflow activities. By discovering conflicts early, the *Scheduling service* is able to better plan future activities.

We need a service that can recognise and remember conflicts. An architectural approach to this was proposed in our depth study [75]. We will present a slightly modified version of that architecture in Chapter 8 and make a proof-of-concept implementation.

The design and further investigation will be dictated by our research questions in Section 2.1.

The implementation starts with scenarios in Chapter 9. These are meant to give us a basis for creating requirements. We then follow up with requirements and architecture in Chapter 10 and 11. After this, we will discuss the choice of technologies for our implementation in Section 12.1. These technologies are presented in Chapter 6 and later evaluated in the discussion (Section 16.3).

Chapter 12 describes the actual proof-of-concept implementation.

CHAPTER 8

Arbitration service

In our depth study we created an idea and an architecture that discovers and handles workflow conflicts in a context-rich heterogeneous environment. This approach is reproduced here with some minor changes. For the original approach we refer to our depth study [75]. Some of the features described in this architecture is not part of the proof-of-concept implementation. This is because we set out to answer our research questions and focus on this.

We start of by describing how we view conflicts in the environment and how they can be discovered.

8.1 Possible conflicts

By arbitration we mean the ability to solve conflicts that may occur. If we look at the scenario in Section 9.1, the clients of the two persons have a goal that is not achievable without cooperation. One will try to raise the temperature by turning the ovens on, and the other one will turn the ovens off to lower the temperature.

Both clients will try to use the heating service for the room. If this service had a schedule, the clients could discover the conflict by the fact that the heating service told the second client that another client is already using the service. This is a preferable way of discovery, and is a conflict based on use of a resources, namely a *resource conflict*.

We use the term *resource*, because it might not only be the use of an actuator that has conflicts. It is obvious that actuators may only support one operation at a time, or over time, hence causing conflicts if more than one entity wants to use it simultaneously. However, other services like sensors, lookup services and so on, may also have restrictions on their use. Sensors may be prohibited from sending results to more than ten clients at a time due to limitations in bandwidth or the update frequency of the requests. This tells us that all resources may be the source of a conflict.

Resource conflicts are fairly easy to detect, but not the only conflicts that may arise.

The trickier conflicts to detect are the ones that are linked to goals, or *goal conflicts*. A *Business process* and a *Process definition*, as they appear in *WfMC*'s workflow definition (Figure 5.19), are *goals* as they only describe what is intended to happen and not witch resources that are used. Resources may change during *adaptive workflow* as described in Section 14.3.

The scenario in Section 9.1 describes first a *resource conflict*, but if we look closer at the case, it really is a *goal conflict*. Both actors intend to change the temperature to different values. Arbitration must therefore check for possible *goal conflicts* when a *resource conflict* is detected.

It is fairly clear that *resource conflicts* may occur without *goal conflict*, but somewhat harder to envision that a *goal conflict* occurs without a *resource conflict*.

If we look at the scenario in Section 9.3, we have such a situation. There, the two forklifts do not share any resources that are in conflict. They both move different pallets to and from different locations, but a conflict occurs anyway. Although the main goal is not in conflict, some of the sub-goals are. The sub-goal of *forklift 1* is to navigate out of the corridor while *forklift 2* has a sub-goal of navigating in. These subgoals are clearly in conflict whilst no resources are, unless you view the actual corridor as a resource.

8.2 Architectural overview

Figure 8.1 shows the architecture of the *Arbitration service* in combination with the *Smart work processes architecture* (Section 6.1). In the figure, we see that arbitration resides within the *Cooperative workflow coordination service* (COWCS) component within the SWP architecture (Section 6.1).

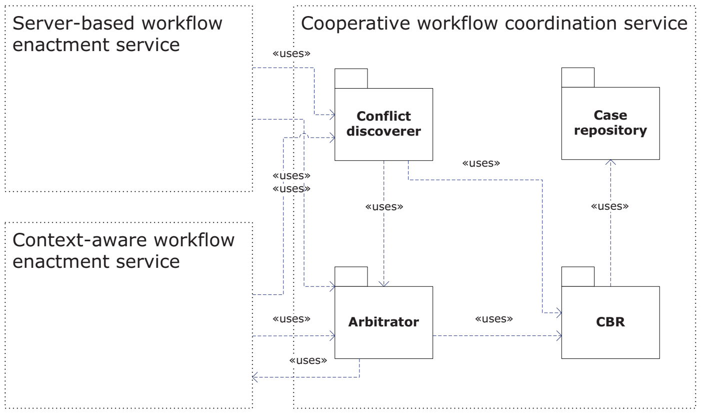

Server-based workflow enactment service
Cooperative workflow coordination service
«uses»
Conflict discoverer
Case repository
«uses»
«uses»
«uses»
«uses»
«uses»
Context-aware workflow enactment service
«uses»
Arbitrator
«uses»
CBR
«uses»

Figure 8.1: Architecture of the arbitration service within *Smart Work Process*.

One of the major components in the arbitration architecture is the *Case repository*. This handles persistency of prior conflicts and their solution and is the repository used by the *CBR* system (Section 5.4.1).

The *Conflict discoverer* component communicates with the *context-aware workflow enactment service* (CAWES), the local *arbitrator*, and the *CBR* system. It will search both local and remote workflows for conflicts.

If conflicts are found, the *Arbitrator* component is responsible for negotiating which entity is to become the conflict handler, ask *CBR* for solutions, getting user input to create new cases, and handling the actual mediation.

8.3 Discovering conflicts

Conflicts regarding resources are commonly found during scheduling of activities. These are then discovered when a schedule tries to allocate resources. The allocation starts with a task querying the resource for usage time in a specific period given by the schedule. It may enquire different periods if alternative executions are scheduled.

A query would return one of the following results:

- The resource is available at the queried time. A degree of certainty would be preferable. If the resource is heavily used or knows the specific period it is particularly popular, it should answer that the queried time-slot, although available, might be taken if it is not allocated immediately.
- Another task has allocated the time-slot, so a conflict must be resolved to possibly be able to use the resource at this time.
- The resource is not available. Most likely this answer would be given if security, authentication, access rights, or other usage criteria are not satisfied.

The scheduling mechanism could deliberately select a plan containing conflicts if it was very advantageous in other ways. If this is done, it is up to the arbitration service to try to resolve it.

The activity diagram in Figure 8.2 shows how the *Arbitration service* gets informed of possible goal-based conflicts locally. Figure 8.3 shows an elaboration of the *Conflict discovery* component.

The basis for discovery of this type of conflict is case-based reasoning. This means that the system will learn by prior experience and use that information to discover when a conflict might arise. If conflicts are found, information about them are stored and conflict arbitration is engaged.

8.4 Arbitration

Figure 8.4, shows an activity diagram depicting the arbitration process for a detected conflict.

The first thing for the arbitrator to do, is to negotiate who get to manage the arbitration. It could be done by one of the conflicting entities, or by a third party in the network. In both cases the clients must supply their knowledge, and user interaction might be required. The figure does not show these steps while arbitration is done remotely.

The first time a conflict is encountered, the CBR solution may seem a bit cumbersome. It will not recognise any conflicts without an existing case-base, and therefore it does not

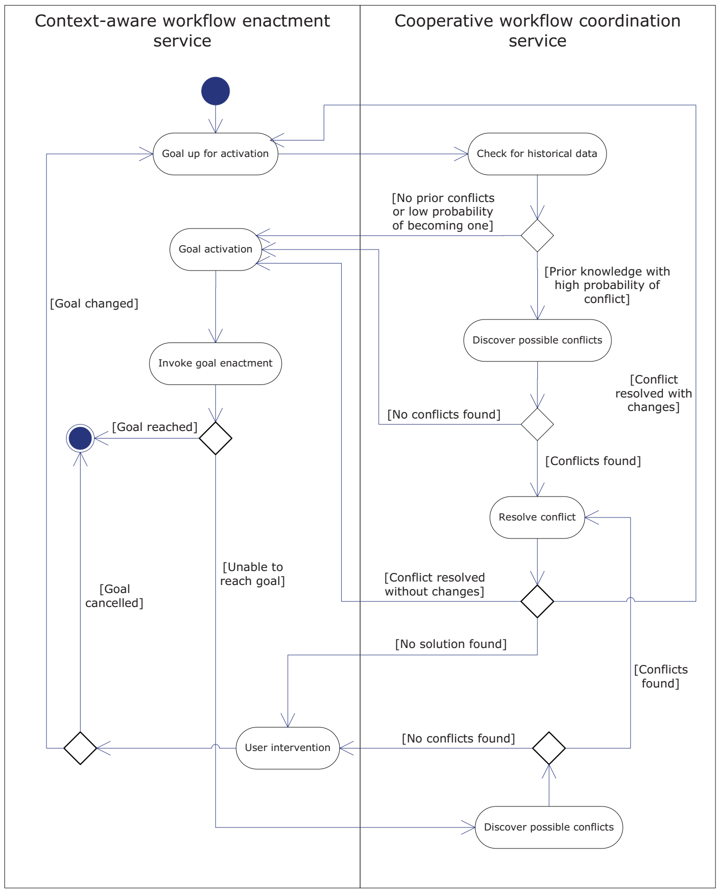

Context-aware workflow enactment service
Cooperative workflow coordination service
Goal up for activation
Check for historical data
[No prior conflicts or low probability of becoming one]
Goal activation
[Prior knowledge with high probability of conflict]
[Goal changed]
Discover possible conflicts
Invoke goal enactment
[Conflict resolved with changes]
[No conflicts found]
[Goal reached]
[Conflicts found]
Resolve conflict
[Unable to reach goal]
[Goal cancelled]
[Conflict resolved without changes]
[No solution found]
[Conflicts found]
[No conflicts found]
User intervention
Discover possible conflicts

Figure 8.2: Activity diagram showing how the *Arbitration service* gets informed of possible goal-based conflicts locally.

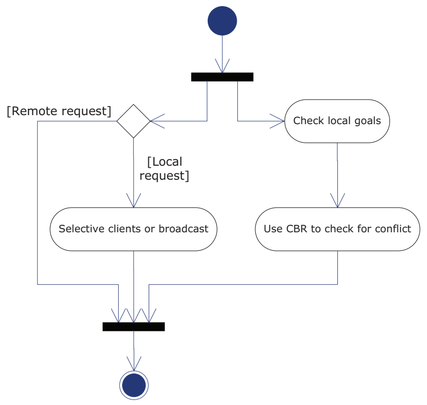


Figure 8.3: Activity diagram showing how the *Arbitration service* checks for conflicts both locally and remote.

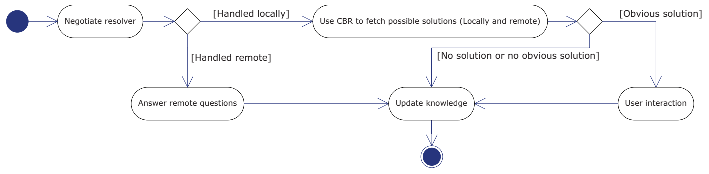


Figure 8.4: Activity diagram showing arbitration when a conflict has been detected.

know how to solve it. In the training of a system, it would be normal for a person to interact and tell the system how to reach a solution in the conflict. As more cases are learnt, the utility and quality of the system will improve.

Ideally, the system could try to port or adapt solutions of other conflicts to resolve the current problem (*RQ 1.2*).

Peers throughout the environment will keep their own case-base as they experience new conflicts. This information must however be shared to gain most benefit from the service. To ensure easy access to all available historical data amongst mobile devices, a CBR proxy could be established. If a stationary, more powerful peer exists in the environment, this could function as a server which holds data conceived in the relevant *Smart space* (Section 3.3).

To decide which entities that should get possible advantages in a negotiation process, weighed scoring could be a solution. Examples of scoring criteria can be found in our depth study [75]. Useful information about the activities causing arbitration can be fetched from the scoring mechanism (which may be part of the activity description), so that a priority can be made.

CHAPTER 9

Scenarios

We have written three scenarios for the *Arbitration service*. Each describes different parts of the planned proof-of-concept implementation functionalities.

9.1 Temperature adjustment

Motivation

This scenario describes the workings of the workflows resource lookup service, how the implementation should discover conflicts that has been found before, selection of a solution, and the user interaction with the system. It is kept as simple as possible to not draw attention away from the important functions of an arbitration system.

Scenario

John is sitting in his office. He is preparing for a meeting with one of his colleagues in the meeting room across the hallway.

As he enters the room, his personal digital assistant discovers that there is a temperature control service in the room. It quickly finds out that John's preferred temperature is 22℃ while the actual temperature in the room is only 20℃. It creates a task to regulate the temperature to 22℃, so that he will be comfortable during the meeting. The small size of the room and the advanced ventilation system quickly adjusts the temperature. It has acknowledged that the desired temperature is within the administrative boundaries set by the service provider.

As he reads through his papers, Lisa, his colleague, enters the room. Her personal digital assistant discovers the service as well, and creates a task to regulate the temperature to her preference which is 24℃. The temperature service acknowledges her right to adjust the temperature, but identifies a conflict between the two requests. As this is not the first time such a problem has arisen, the system sees that the normal solution to this problem is to join the two tasks and calculate the mean between the desired temperatures. An informative text shows up for the task in the personal digital assistant of both indicating that a compromise has been reached. The information is given, but no visual or audible alert is provided due to the commonness of the solution. If the service had been in any doubt about the solution, it would have given a notification to the participants and possibly halted the compromise, pending interaction.

The system then adjusts the temperature to 23℃, only to regulate it down to idle temperature when they leave the room.

Challenges in scenario

- Separate conflicting tasks from those who can be performed in parallel.
- Reach an agreement on parameters when merging of tasks is needed.
- Automate as many of the decisions as possible to prevent the user from having to interact all the time.
- Making sure that tasks are within the administrative boundaries of the service. This may be minimum and maximum temperatures, how fast to regulate temperature, if their stay in the room is of such length that it serves a purpose, security concerning who are allowed to be part of it, quality of data, and the possibility to pre-adjust the temperature to their preferences before they arrive at the room.
- The use of fuzzy sets instead of crisp values would make these kind of transitions smoother and more dynamic, but also more complex to implement.

9.2 Humidity adjustment

Motivation

As the scenario in Section 9.1, this scenario is very basic. It is only intended to depict the narrow problem of knowledge transfer between domains. This scenario is a direct case where the system is able to use a general domain knowledge network to detect a conflict not before discovered based on a prior discovered one.

Since John's digital assistant already knows how to handle the temperature conflict in Section 9.1, he follows us also in this scenario.

Scenario

After work, John goes to the gym. He usually goes there after work, since there are fewer people there then. Most of his colleagues goes there before work, but John likes that most of the equipment is free for use.

After changing, he enters the equipment room to start his work out. Susan, the accountant, is performing yoga while John steps up to the treadmill. Susan's personal assistant has been adjusting the temperature and humidity of the room to her likings, but now that John enters, two conflicts arise. Since Johns assistant knows how to solve the temperature conflict, this is resolved only with a silent information message on the display. Now John likes the humidity of the room to be fairly high because he does not like to get a dry throat when he runs, Susan on the other hand, likes it fairly dry as she does not want to get clammy doing her yoga. Neither of the digital assistants have ever seen a conflict like this before, but Johns assistant quickly suggests using the same solution to this problem as to the temperature due to their similarity of nature. John says that this seems to be a great solution, selects it, and enters that; in the future, if something is this similar, do not ask me, just solve it.

When Frank enters the room, his digital assistant is automatically brought up to speed by Susan and Johns, and they all work out in an environment of their liking.

Challenges in scenario

- How to automatically discover conflicts that have not been discovered before.
- If a possible conflict is discovered, how can the case or solution be adapted. Especially regarding complex solutions using fuzzy sets or concrete knowledge.
- Automate as many of the decisions as possible to prevent the user from having to interact all the time

9.3 Warehouse conflict

Motivation

The two prior scenarios have been very primitive. The workflows are very basic, as many things are in real life, but often the conflict are more complex. This scenario is intended to give a more practical use of an *Arbitration* and *Scheduling service* (Chapter 15) in a work situation.

Scenario

PalletStorage Ltd. is a company handling intermediate storage of pallets. They have for a great deal of time looked for ways to automate the handling of pallets to save money and make shipping and storage more efficient.

Last week, two automated forklifts were delivered. These forklifts will fetch pallets and place them at designated locations. These locations are governed by a server keeping track of which pallet is to be placed where.

One day the two forklifts, conveniently named *forklift 1* and *forklift 2*, where to place pallets besides each other in a narrow corridor.

As *forklift 1* placed its pallet in the end of the corridor, *forklift 2* recognized that *forklift 1* was already there, and that the corridor was to narrow to pass each other. Having been in a similar situation before, *forklift 2* waits until *forklift 1* has exited the corridor before entering.

The first time they were in a situation like this, *forklift 2* followed after *forklift 1* into the corridor. This resulted in that *forklift 2* blocked *forklift 1* from getting out and *forklift 1* blocked *forklift 2* from placing its pallet, ending in neither getting their work done. The

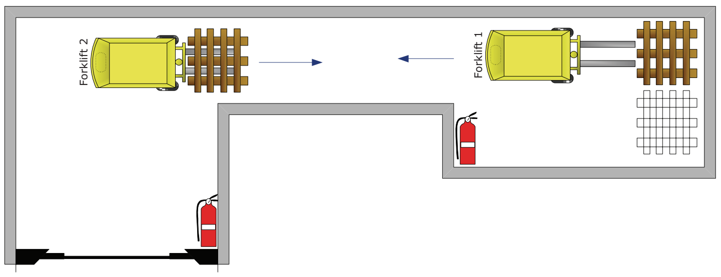


Figure 9.1: Showing forklift conflict in warehouse with driving patterns and placement of second pallets.

situation was solved by *forklift 2* backing out, letting *forklift 1* pass it, before entering again.

Forklift 2 waited outside because it anticipated that a conflict could occur, having been in the situation described above, it asked *forklift 1* if they might end up in a conflict if it entered.

This way the forklifts cooperate and foresee future problems that may occur, completing the work in an efficient way.

Another solution, the forklifts have not learned, is that *forklift 2* could enter and place the pallet near *forklift 1*, leaving it to this to do the job. This way the forklifts could cooperate on moving several pallets into an awkward position, not having to wait on each other exiting the corridor and hence being even more effective.

The supervisor aims to tell the forklifts, the next time they enter a situation like this, to use that solution, rendering them capable of taking that decision in the future.

Challenges in scenario

- How to find conflicts regarding what is intended to be done, without any conflicts with location of pallets and no pre-stored data about the situation.
- To store a solution and reuse it when a similar situation occurs.
- If several solutions are found, which is the best? The one where *forklift 2* waits or the one where it leaves the pallet for *forklift 1* to place.

CHAPTER 10

System requirements

We have decided to develop a vertical prototype of a workflow enactment system with focus on the *Arbitration service* described in Chapter 8. For this to work, we also need a *context-aware* platform for the services to run on. The following sections will outline the requirements for the system.

10.1 Communication requirements

The first thing the system must support is communication between peers. This requirement is critical for running and testing the implemented services, as they are supposed to work not only locally on clients, but also between clients. The platform must support an open communication standard to ensure that all kind of peers can take part in the network, without limitations caused by proprietary and non-interoperable standards. This forms the functional requirement **F1**.

Another communication requirement is that connections between peers needs to be flexible, allowing clients to enter and leave a network as they please. This requirement, **F2**, emerges from the volatile nature of ubiquitous computing, where heterogeneous equipment enter and leave network coverage area and need to establish connections to be able to communicate and cooperate.

Sharing of context information, workflow schedules, arbitration information, sensors, actuators and other information between clients requires that all components are addressable. With this, a client can discover if the remote client is running a specific component and that communication can be routed directly to one component on another client. This is our requirement **F3**.

Since **F2** dictates that the network is of ad-hoc fashion, the components of the implementation should not, unless strictly necessary for the functionality of the component, be dependent on a network connection being present (**F4**). Especially is this important for workflow enactment components.

10.2 Context requirements

The system's utilitarian value is decided by its ability to respond to a dynamic environment. A workflow is dependant on context information to execute and the arbitration and planning services need context to make the right decisions. The system needs to be context-aware (**F5**). See Section 5.1 for more information on context-awareness.

Requirement **F6** states that contextual information should be shared in the peer-to-peer network, enabling other clients to look up information relevant to them. Forwarding of all context will be too resource demanding and serve little purpose, so the peers should collaborate in building a "world context".

To avoid having several communication technologies, the *Context service* should use the same communication interface as the rest of the application. This forms requirement **F7**.

No handling of sensitivity, confidentiality, validation, correctness, accuracy or time-since-update is necessary in this implementation.

10.3 Resources requirements

The system must support sensors, actuators and information sources. These are from here on called *resources* (**F8**). Resources should, as with context, use the implemented peer-to-peer communication interface which form the single point of communication (**F9**). This is to ensure that the communication implementation is reliable and support interoperability. Requirement **F10** states that all resources running on a client should be searchable and accessible from all clients in the network. This is to support an actual environment and enable cooperation.

10.4 Workflow enactment requirements

As a method for testing the arbitration component, requirement **F11** states that the system must be able to enact basic workflows. The complexity of the workflow enactment component should be kept at a minimum, but it must support functionality as *start*, *pause*, and *stop* (**F12**). As the inclusion of *start* and *stop* is fairly obvious, *pause* is justified as a probable part of an arbitration solution.

Solving a conflict regarding a workflow might involve adapting it or joining two or more together. The system must therefore support dynamic changes to workflows, **F13**, and monitoring or connecting to a remotely executed workflow, **F14**.

For the workflow enactment to be of any value, it must be able to use context and resources during execution (**F15**).

10.5 Arbitration requirements

The arbitration component is the base of our proof-of-concept implementation. Our architecture is based on CBR, and there are some requirements in connection with this.

First of all, the it must be able to recognise a possible conflict from prior learned cases

(**F16**). When new cases are entered or learned, these must be saved to a persistent storage (**F17**).

When searching through the workflow context for possible conflicts it must be able to sort any hits according to relevance. Requirement **F18** describes this, which is important to select the case best matching the current situation.

If a prior conflict is found but it is only similar, it must be able to adapt (**F19**) this case so it can be stored in the repository. If no prior cases are found for a conflict the user must be able to enter a desired solution, **F20**. Both **F19** and **F20** enables the system to learn from either experience or input.

The arbitration service must be able to detect conflicts, not only amongst local workflows, but most importantly between itself and other peers (**F21**). It must therefore be able to communicate, using the single point of communication, with other clients to search for possible conflicts.

It may not always be desirable that a local client performs the arbitration. It may want to leave this responsibility to the other client or a third party. The negotiation of who will be responsible for performing arbitration is stated as requirement **F22**.

As is possible with context and resources, the arbitrator must be able to draw knowledge from the other peers in the network to help it perform arbitration, **F23**.

After a new conflict is discovered or during arbitration, the case representing this conflict and solution should be passed to all parties (**F24**). This is to enable clients to learn from each other. A client may detect a conflict which is known to it but not to the other peer.

For the arbitration service to actually be able to perform arbitration, it must be able to remove, add, or adapt workflows in the workflow enactment service, (**F25**).

10.6 Functional requirements summary

Communication

F1	Support an open communication standard, i.e., not be bound by hardware or proprietary standards.
F2	Support ad-hoc, in-situ connecting and disconnecting of clients.
F3	Software components on a client should be addressable from the network.
F4	The software should not be dependent on a communication service always being present.

Context

F5	The system needs to be context aware.
F6	A client should share context with other clients.
F7	The context client should use the communication component.

Resources

F8	Incorporate sensors, actuators and other knowledge information.
F9	Should use the existing communication component.
F10	Distribution of resource information and/or discovery of resources in the environment.

Workflow enactment

F11	Enact basic workflows.
F12	Handle start, pause, stop of workflows.
F13	Handle dynamic changes to workflows.
F14	Monitoring of workflows executed remotely.
F15	Read and manipulate resources.

Arbitration

F16	Find similar conflicts in the repository.
F17	Persistent repository.
F18	Order similar conflicts according to the degree of similarity.
F19	Adapt prior solutions to new conflicts and store them in the repository.
F20	Call for solution if none is found.
F21	Be able to discover conflicts between other clients and itself.
F22	Negotiate which client is responsible for the arbitration.
F23	Make use of other clients knowledge.
F24	Distribute solution to participants so that they also learn.
F25	Must be able to remove, add or adapt workflows in the workflow enactment component.

10.7 Non-functional requirements

R1	Separation of concern. Functionality must be clearly separated in the architecture.
R2	The architecture must be platform independent.
R3	Response times must be within reasonable and expected limits. If this is not achieved, information must be displayed.
R4	Because this is a research implementation the architecture must be easy to evaluate and understand.
R5	It must have a high level of maintainability.
R6	Loose coupling between components.

CHAPTER 11

Architectural description

This architectural description is based on the *IEEE Recommended Practice for Architectural Description of Software-Intensive Systems*, IEEE Std. 1471-2000 [47].

It is a result of the arbitration architecture proposal from Chapter 8 and our prestudy [75], and is the guideline for our proof-of-concept implementation.

11.1 Architectural drivers

The architectural drivers of our vertical proof-of-study implementation are as follows:

Separation of concerns (SoC)

It is important for the architecture to clearly separate the different aspects of the implementation. The arbitration part must be clearly separated from workflow enactment, context, network, the graphic user interface, and other parts of the application.

This will greatly help us develop each component by itself and keep an iterative development process.

It is also a great contributor for easy evaluation and traceability of the implemented functions as well as helping with problem discovery.

Modifiability

Components of the implementation must be easy modifiable. Bearing in mind that this is a prototype implementation, it must be readily able to test different implementations of components.

It might be that one of the implemented resource does not perform as expected. It might then be helpful to create another, slightly different, implementation and easily exchange these. Some components of the system, namely services, should be so loosely coupled that they can be exchanged runtime.

Testability

The system must, in an extension to modifiability, also provide testing options and good diagnostics information. This is important to ensure that all components deliver results as intended.

The architecture must support monitoring of selected components and maintain diagnostics information.

Accountability

Since this is a decision support utility, it is very important that the system can account for all decisions it makes. In our case, it is most useful to test that the components perform as expected.

If there is any unsuspected behaviour, why did it go wrong? If the CBR detects a conflict between two workflows, what were the deciding factors? How did it reach a conclusion that a new situation resembled an already known case to a degree that it constitutes a conflict?

All these questions must be readily answered by the implementation to be able to test that it behaves as expected or wanted, and so that any unsuspected behaviour can be recorded and analysed.

11.2 Stakeholders and concerns

There are no intended end users of this architecture. It is part of a master thesis as a proof-of-concept implementation. The stakeholders are therefore the master students and teaching supervisors.

Master students

- Indahl, Christian
- Rud, Kjell Martin

Teaching supervisors

- Sørensen, Carl-Fredrik
- Kofod-Petersen, Anders

11.3 Architectural patterns

The most prominent patters are descibed below. In addition to these, there is a widly use of the basic control patterns *Sequence* - execute activities in sequence, *Parallel Split*

- execute activities in parallel, and *Synchronisation* - synchronise two parallel threads of execution.

Blackboard pattern

We will have a framework-class that serves as a blackboard where all components can get hold of an instance of another component. The components available from the framework-class will be log, network, arbitration, ontologies, context, workflow subsystem, and graphic user interface (GUI).

This is chosen because it is built as a block system in an incremental way. By using this acrhitectural pattern, each block can get access to another block directly from the blackboard and it radically reduces the maintanence demands if changes to the architecture were to occur.

This helps in realising the *SoC* and *Modifiability* architectural drivers.

Control loop pattern

This pattern will be used by almost all types of monitors in the implementation.

Services that monitor peers, context and resources will utilise this pattern as they will need to constantly have an updated list of peers and search for new conflicts.

It will also be an important part of arbitration and workflow enactment. The arbitration service will take advantage of this pattern to search for possible conflicts and in workflow enactment it will be used to check incentives for and for monitoring of remotely executing workflows.

Exclusive choice pattern

This will be a widely used pattern. A choice on execution path will be made from several alternatives based on input.

Typically this would be a workflow with a conditional statement. If we view the workflow execution path as a branch forking, where each forked branch executes a different set of statements, and then joins, exclusive choice means that only one branch will be selected each time.

Observer pattern

It defines a way for classes to be loosely coupled and for one class (or many) to be notified when another is updated. Basically, this means that when something happens in one place, you notify anyone who is observing.

This pattern will be found throughout the implementation. It will be the main way to trigger the different actions. If the *Conflict discoverer* finds a possible conflict, it will inform of this through the observer pattern implementation.

Since a loose coupling is highly desirable, this will also be the only way of interaction between the GUI and the business logic.

11.4 Views

Uses view

Figure 11.1 shows the top level uses view for our architecture. The idea is that the components are primarily independent of each other, in accordance with *SoC*.

All communication between the user and the software will be done through the *GUI* component. No interaction or display of values will be done by the other modules. To achieve this, the components must use the observer pattern to tell the *GUI* of changes or input needed.

The *Log* component will maintain a log of all infomation, supplied with source and level, that is of interrest. The different levels are information, debug, error and fatal error. It is the logs responsibility to filter out any messages.

The *Network* component will support the application with all communication needs. It will create a network connection, and inform, using the observer pattern, interrested parties when it is connecting, connected, disconnecting and disconnected.

It will also help other components to communicate or establish connections with other peers.

Searching all peers it finds, the *Arbitration* component will try to discover possible conflicts between the local workflow enactment and the remote peer. If it finds any, it will engage in an abitration process. Conflicts will be searched using CBR and ontologies.

The system will use ontologies to classify different type of workflow processes, solutions to conflicts, and context sources. Because of this variety of use, the *Ontologies* component will be made available from the framework.

For the arbitration process to succeed it must be able to see which workflow processes that are planned or executing. Along with other components needing to create or update workflows, we found it useful to make *Workflow enactment* accessible from the framework.

The *Context* subsystem will be available from the framework blackbord class. The workflow enactment and the arbitration service will depend on contextual information, sensors and actuators. This renders the component as a high in demand component which justifies it as a blackboard component.

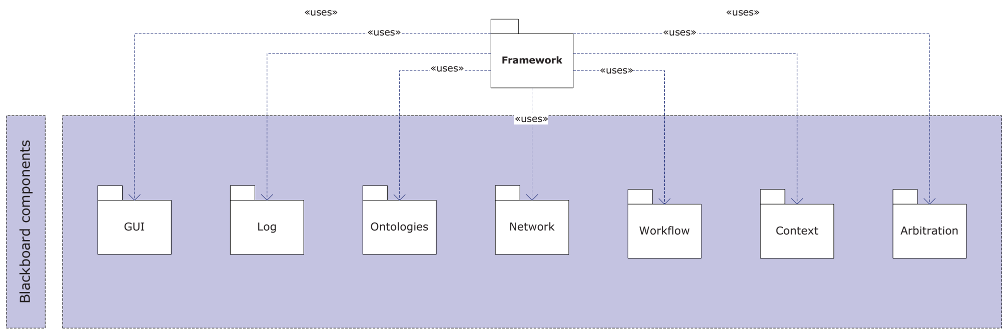


Figure 11.1: Show the uses view of the architecture.

Process view

Figure 11.2 depicts the arbitration process.

First, a search for peers is done. If no peers are found, it will wait and try again later. For each peer that is found, it will search for possible conflicts. Then the following happens:

- If no conflicts are found; wait a configurable period and restart the process of searching.
- If an obvious confict is found; Start the arbitration process on involved workflows, wait a configurable period, and restart the process of searching.
- If a possible conflict is found; Ask the user to select an appropriate action. This could be to ignore it or to adapt the solution from the possible conflict to solve this specific one. If the latter is chosen, it will adapt and learn the case. This will then later appear as an obvious solution.

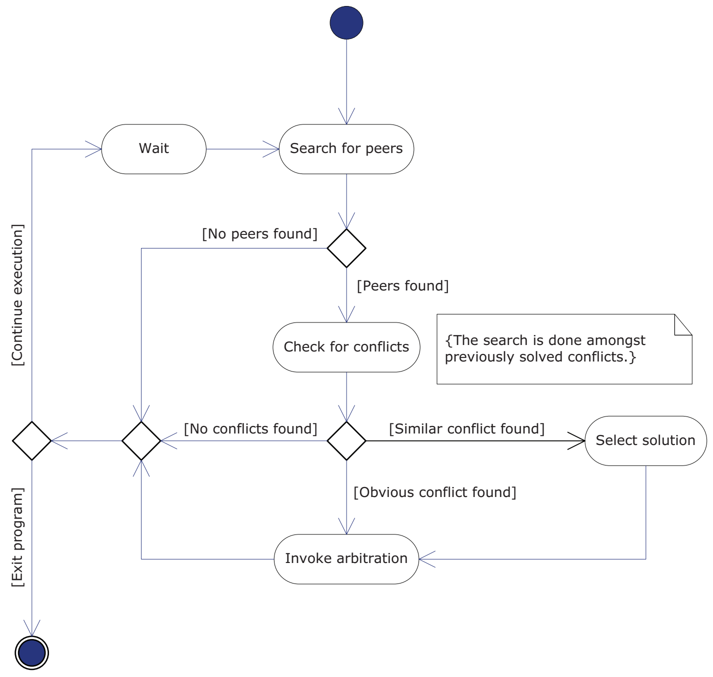


Figure 11.2: Show the arbitration process.

CHAPTER 12

Implementation

Our proof-of-concept implementation has the following capabilities:

- Forms a peer-to-peer network with other clients running this implementation.
- Running sensors, actuators, and contextual sources, letting all clients on the network use them. These are here on called resources.
- Create and execute workflows that can utilize resources.
- Identify conflicts new occurences of known conflicts.
- Predict new possible conflicts based on resemelance with known, adapt and store these. A new case is then learned.
- Perform basic arbitration on running workflows.

In total there are 233 classes and interfaces origining from 123 java files with 12,215 lines of code, but only a few major and important classes are covered in this implementation description.

12.1 Choice of technology

We chose JXTA as our communication component. JXTA is the industry-leading P2P technology, supported by over 30,000 members worldwide with downloads exceeding 12 Million [30]. It support several different communication technologies as Bluetooth, wireless, IrDA, wired, and others. Implementations of JXTA is created for C/C#, J2SE, and J2ME. This makes it a powerfull tool.

Since our implementation is suppose to work in heterogenous evironments, and support ad hoc mobile work, a peer-to-peer solution was required. JXTA was therefore a perfect solution for our proof-of-concept implementation.

Our implementation needs to be context-aware. After considering several frameworks the choice fell on JCAF. JCAF, or Java Context-Aware Framework, is created by Jacob E. Bardram as a closed source project. Supporting peer-to-peer context service, implemented in Java, support for contextual listeners, and easy implementation of context subscribers and consumers, it fell as a natural choice.

We needed to find a CBR system that could be integrated with our implementation. Since the other technologies were implemented in Java we needed to find a CBR system that also supported this. The choice fell on jCOLIBRI which is a mature KI-CBR framework, supporting customization of almost every part of the CBR process.

Our proof-of-concept application also needed to implement ontologies. A technology supporting this is Jena. It ships with jCOLIBRI distribution and is instanciated and used by jCOLIBRI if the description logic extension is used. We do therefore not need any other implementation than this, so Jena is the choice for ontologies.

12.2 CIKMR package

This is the base package for the proof-of-concept implementation. The important classes of this package are *Framework* and *XMLRepresentation*.

Framework is based on the blackboard design pattern. It provides access to the important components of the application. *XMLRepresentation* is an interface implemented in all classes that has an xml representation.

12.2.1 Framework

This is the class implementing the *Blackbord pattern* from the *Architectural Description*, (AD), in Chapter 11.

Figure 12.1 shows the components provided by the Framework class. It also shows which components that are services and resources. What constitues a *service* and a *resource* is covered later on together with a more detailed explanation of each component.

There are two main methods of the Framework class, *startup* and *shutdown*. These are the methods that are invoked to start and stop the application. They will handle all object instanciation and destructions. This is done through method invocations and by firing events. An activity diagram of the startup and shutdown process is provided in figure 12.2.

12.2.2 XMLRepresentation

The interface for all classes that can be represented by *XML*. It defines methods for creating an *XML* representation of a class instance and to restore the state of an instance from an *XML* representation.

It is used in a wide variety of classes. The complete set of statements, activities and goals for workflow enactment, representation of conflict searches, solutions and other classes.

The different *XML* schemes will be presented together will the classes or activity they represent.

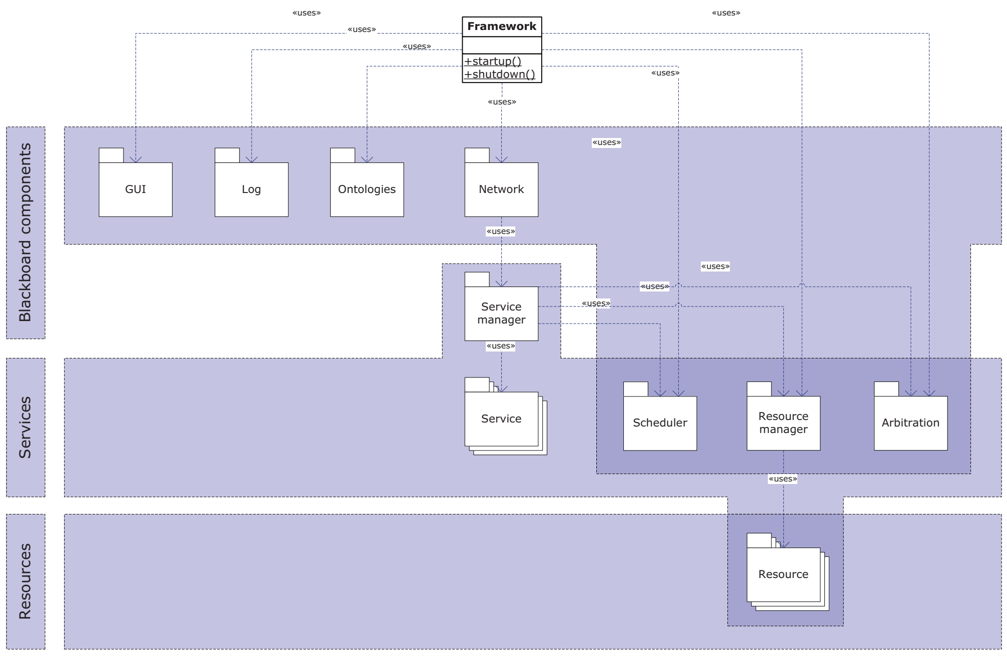


Figure 12.1: Shows the blackboard design pattern of the Framework class in the CIKMR package.

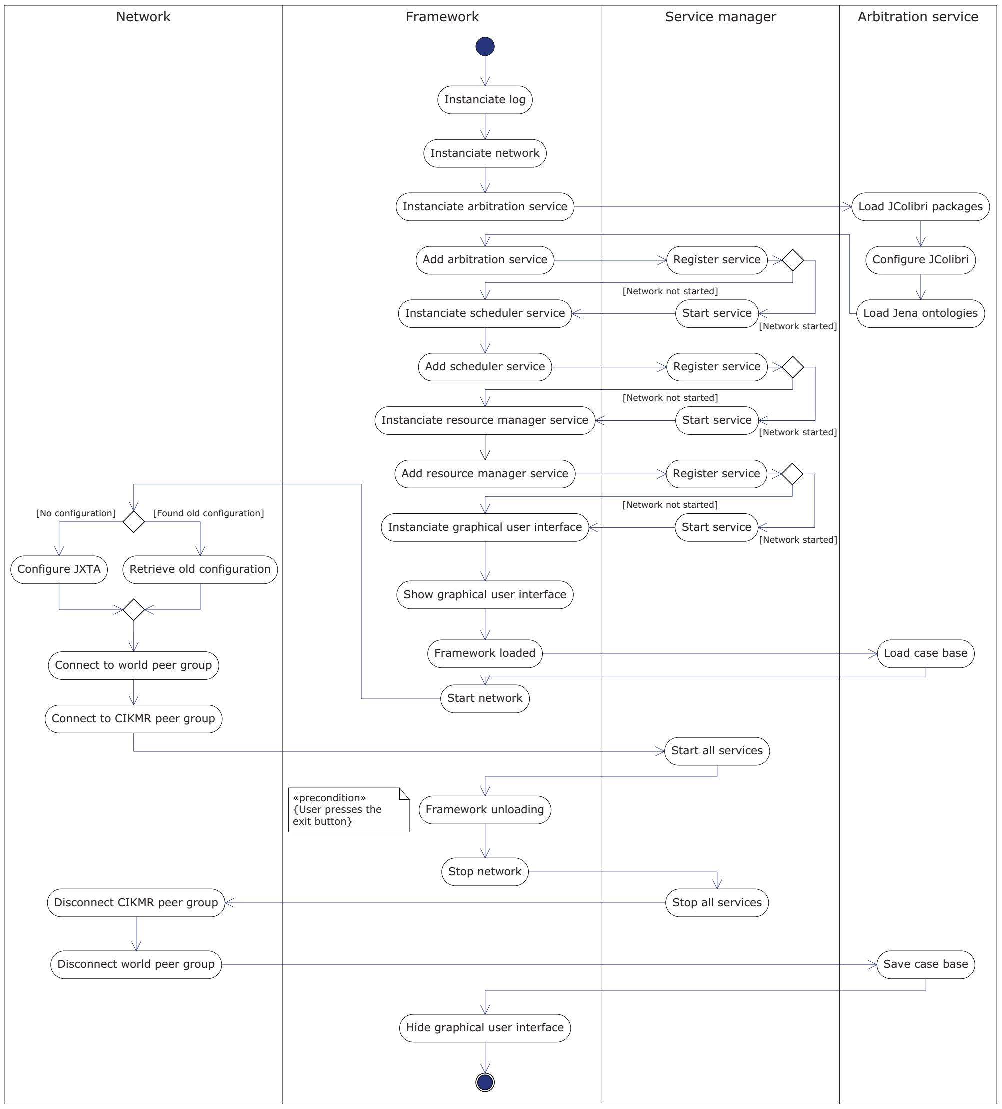


Figure 12.2: Shows the startup and shutdown process of Framework class.

12.3 Network package

This package is responsible for the applications network connection. An activity diagram of the network lifecycle is provided in figure 12.3.

It has three major classes. These are *Network*, *PeerGroupConnection*, and *ConnectionStatus*. The *Network* class is accessible from the *Framework*, as shown in Figure 12.1, and is the accesspoint of the applications network. *PeerGroupConnection* is the class responsible for configuring*JXTA* and connecting to the peer groups. *ConnectionStatus* holds status information on the connections.

12.3.1 Network

This class is the access point and the manager for the applications network connection. It creates an instance of the *PeerGroupConnection* class and has a *connect* and a *disconnect* method.

The other important part of this class is that it creates and holds an instance of the *ServiceManager* class. It informs this instance when the network is connected or disconnecting. The *ServiceManager* class will be described later.

12.3.2 PeerGroupConnection

The application uses JXTA for network transport and is therefore part of a peer-to-peer network. This class is responsible for configuring JXTA, setting up the peer groups and connecting to them. An illustration of this is found in figure 12.3 depicting the network lifecycle.

It starts by checking for existing configuration. This configuration is only present if the application has been run before and the configuration was successfully stored. If found, it will load it and continue, if not, it will create a new configuration and ask for display name to be entered. After this, it will first connect to the *world peer group*. This is the default peer group that all jxta clients must be members of. Second, it will connect to a specific peer group created for this project.

No *rendezvous peers* or *relay peers* are used in this setup due to the small scale of this proof-of-concept implementation. Hence the peer-to-peer network that evolves when running the application is based only on *edge peers*.

12.3.3 ConnectionStatus

An instance of this class holds the status of the connection. It will inform all listeners when the connection enters the different stages or of errors that occur. The connection stages are:

- Disconnected
- Connecting
  - Configuring network
  - Connecting to world peer group

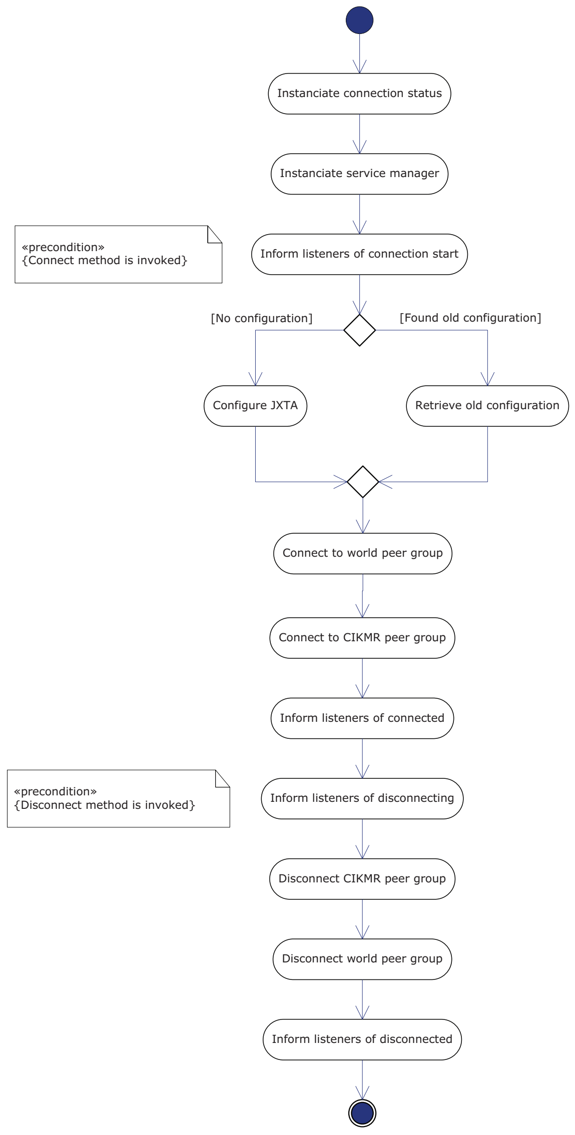

Instanciate connection status
Instanciate service manager
«precondition»
{Connect method is invoked}
Inform listeners of connection start
[No configuration]
[Found old configuration]
Configure JXTA
Retrieve old configuration
Connect to world peer group
Connect to CIKMR peer group
Inform listeners of connected
«precondition»
{Disconnect method is invoked}
Inform listeners of disconnecting
Disconnect CIKMR peer group
Disconnect world peer group
Inform listeners of disconnected

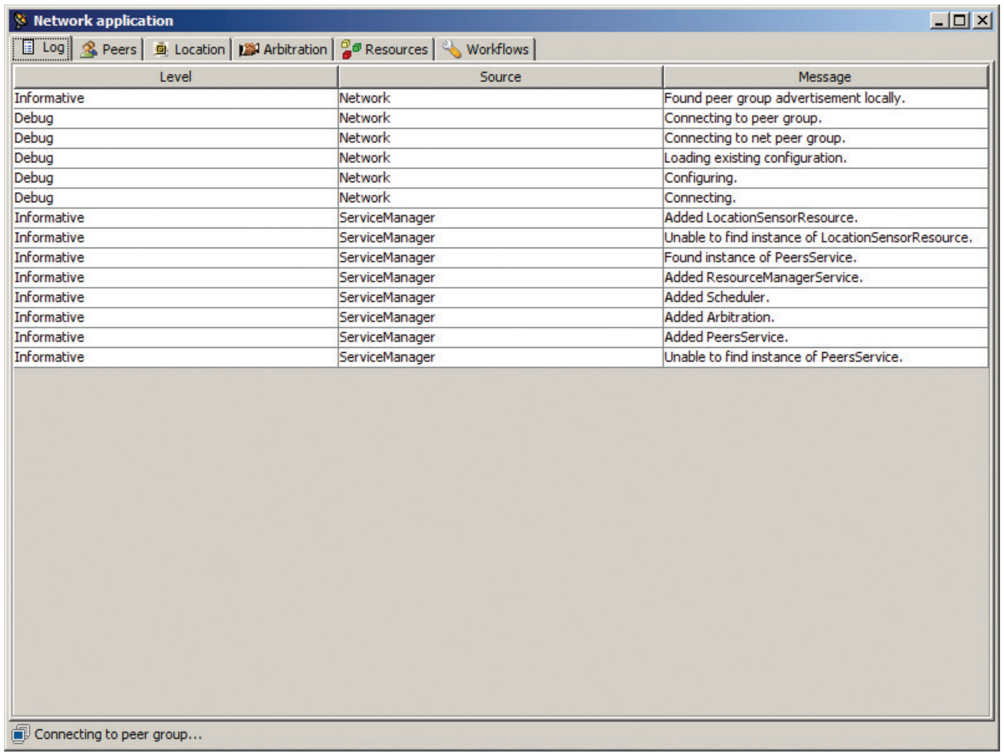
Figure 12.3: Shows the lifecycle of the applications network component.

  - Connecting to project peer group
- Connected
- Disconnecting

The error classifications are:

- Unclassified
- Timed out
- Error connecting to world peer group
- Error connecting to peer group

Figure 12.4 shows a screenshot of the implementation connecting to the project peer group. The current stage of the connection process can be seen in the status bar in the lower left corner. From the log, we can see that it found the configuration on persitant storage and used that instead of creating a new. The log is read bottom-up.



Level	Source	Message
Informative	Network	Found peer group advertisement locally.
Debug	Network	Connecting to peer group.
Debug	Network	Connecting to net peer group.
Debug	Network	Loading existing configuration.
Debug	Network	Configuring.
Debug	Network	Connecting.
Informative	ServiceManager	Added LocationSensorResource.
Informative	ServiceManager	Unable to find instance of LocationSensorResource.
Informative	ServiceManager	Found instance of PeersService.
Informative	ServiceManager	Added ResourceManagerService.
Informative	ServiceManager	Added Scheduler.
Informative	ServiceManager	Added Arbitration.
Informative	ServiceManager	Added PeersService.
Informative	ServiceManager	Unable to find instance of PeersService.

Connecting to peer group...

Figure 12.4: This screenshot shows the implementation connecting to the project peer group.

12.4 Services package

This is one of the most important packages of the application.

A service is a software component that is dependent on an existing network connection. As an alternative to letting all individual components listen to the *ConnectionStatus* component of the *network* package, a *service* is started and stopped by the *ServiceManager* when a network connection is established or lost.

It also serves another important purpose. Since this application should be easily extendable, it needed a way to grow without changing to many classes or interfaces. By incorporating services, we have given the application the means to grow by just adding new services at run-time. All services are given access to the network.

Any component implementing a service is meant to be part of the application itself and not part of any context sources, sensors or actuators. A higher evolved version of a service, called a resource, is implemented for these. It will be covered in detail in Section 12.5.

This package has the *ServiceManager* class which has the repsonsibility of managing all services at runtime, a *Service* interface defining a service, and an *AbstractService* implementing listener methods. In addition there are some helper classes.

12.4.1 ServiceManager

The workings and some of the responsibilities of this class is covered in both Figure 12.1 and 12.2. It is a very simple, yet important class. One instance is created and managed by the *Network* class. During run-time is holds and manages a list of all registered *Services*.

When the network is connected, it starts all registered services. When the network disconnects it stops all services before the disconnection happens.

It also defines methods to register and unregister services and lookup methods. These lookup methods include type-wise lookups where you can ask the manager for a specific type of service. This is meant to prevent components to instantiate several instances of the same service. One example of this the PeerDiscovery service, Section 12.4.3. It is not an essential part of the application and need therefor not be a static part of the implementation. It is added by the components that need it, in present case three components, but they do not know in which order they are instantiated or about the others that may need it. The correct paradigm to use when a service is needed is therefor to first check if the service exists, if not, instantiate and add it.

12.4.2 Service

This is an interface defining a service. Most essential components in the implementation use this. It has three methods that are called during the different stages of the lifecycle.

init(ServiceManager s) This method gets called once, and only once, on each service. It is called when the service is registered with the *ServiceManager*.

start(PeerGroup p) Gets called on every registered service when a network connection has been established, or right after *init(ServiceManager s)* if a network connection was already established upon registering.

stop() Called when the service is unregistered while connected or when the network is disconnecting.

AbstractService is an abstract class that implements add, remove and fire methods for the *ServiceListener* interface.

12.4.3 Peer Discovery Service

This is a service that periodically scans the network for changes in peer composition. It will maintain a list of the peers it currently knows about, adding new when found and removing those it lost contact with.

Application components can use this service to list and communicate with peers. It will also inform interested parties when a new peer is found or when contact has been lost and the peer is removed from the list.

Figure 12.5 shows how this service works. The discovery service of JXTA is asynchronous. The three paths are therefor one thread that sends broadcast messages to the network requesting peer advertisements, one thread removing peers from the list that has not given any sign of life for a while, and a listener that receives the responds from the other peers and updates the list.

12.5 Resources package

This package was first intended to support sensors and actuators. A resource is here defined as a smart entity from which you can invoke advertised methods on. This is almost like RMI over JXTA but with advertisement of the methods, parameters and return values.

Since this application needs to be context-aware, we looked for existing context software to implement. The choice first fell on Java Context-Aware Framework (JCAF) developed by Jacob E. Bardram. When implementing it, we found that it could not communicate over JXTA but used RMI with direct IP addressing of other context servers. Since this breaks with our peer-to-peer design, workarounds and other solutions were considered.

Our application does not need complex context representation to perform its task as a proof-of-concept for arbitration using CBR and ontologies. Our choice was therefor to use resources as our context framework. Although resources does not fulfil most well known paradigms of context modelling, like Henricksen et al. [50], it will serve our needs the best. It also means that we will save time on our implementation. This by avoiding having to implement complex workarounds and context integrity systems. Furthermore, it will ease the implementation of workflows, leaving us with only one type of information and actuation source.

This package therefor represents sensors, actuators, context and any other resources needed by our implementation.

The client always communicates with a resource over JXTA even though it is created, executed, and used on the same client.

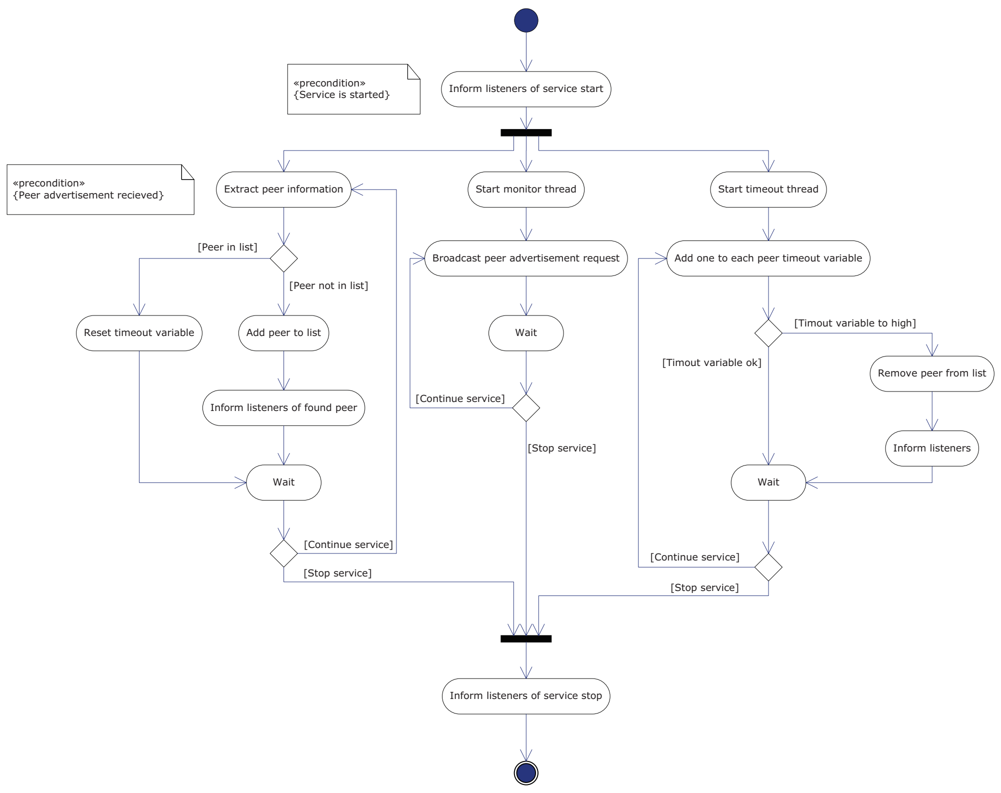


Figure 12.5: Shows how the peer discovery service works.

12.5.1 Basic manner of operations

A resource has one class, *AbstractResourceService*, that is the base of the actual resource. By extending this class you can make a sensor, actuator, context, or information source.

A resource can be used from any client. To allow this, a remote snippet of the resource, namely the *Resource* class, is instanciated on each client that discovers the resource. This snippet holds information on what type of resource it is, what methods it has, and lets the client invoke methods on the remote resource. These snippets are the only thing the clients need to know about to use the resource. The actual implementation or location of execution is irrelevant. An illustration of this can be seen in figure 12.6. More detailed information on each component of resources is described further on.

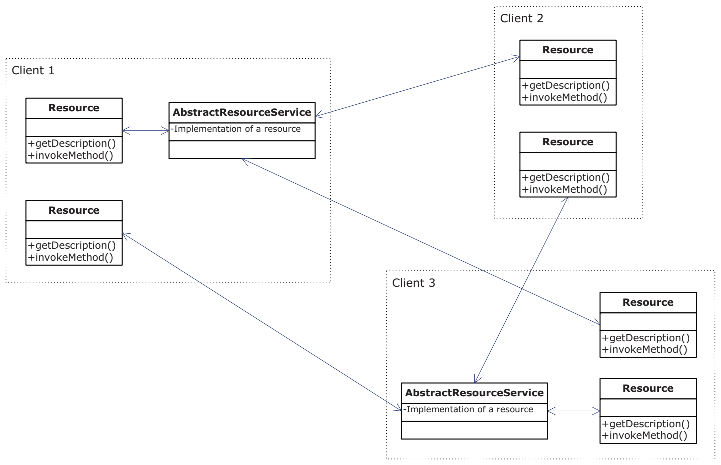


Figure 12.6: Shows where resources are executed and how they are used in a network of clients.

There are four major classes of this package is the *ResourceDescription*, *AbstractResource-Service*, *Resource*, and *ResourceManagerService*.

12.5.2 ResourceDescription

This is a meta data class that describes a resource. An implementation of a resource must specify this description.

The following things are contained within the description:

- Descriptive text of the resource.
- The ontology individual that describes the type of the resource. This is referred to as type elsewhere in this document.
- The ontology individual representing its physical location.
- A list of methods that the resource supports. For each method the following things are described.

- The method name
- A list of parameters that must be passed. This list consists of the parameters data types in order of appearance.
- The data type of the return value.

All data types can be used. New data types can be registered run-time to add support for new types. Currently int, double, boolean, String and void is implemented.

This description is converted to XML by the *AbstractResourceService* and sent to peers requesting it. It is then converted back to a *ResourceDescription* hierarchy in a *Resource* instance on the requesting peer. Within the *Resource* instance, address information to the peer implementing the resource is held. The resource is then ready to be used.

The *DTD* and example *XML* can be found in Appendix A, while Appendix B show how to create a resource description in Java.

12.5.3 AbstractResourceService

This is the class to extend when writing sensors, actuators and context sources. It is based on the *AbstractService* class and it therefore inherits from the *Service* interface.

To create a resource, all you have to do is to create the *ResourceDescription*, as explained above, and to implement the methods you added in the description. To start and stop the resource, register and unregister it with the *ServiceManager*. The rest is handled by the implementation. It will create an JXTA server pipe, advertise it, and wait for clients to connect and use the resource.

Clients can either ask for the description or invoke methods. If the description is asked for, it will send the *XML* variant of the resource description. If a method invocation request comes, it will find the method in the description, look up the actual method using Java reflection and try to invoke it with the supplied parameters. After invocation it will return the result to the invoking client.

Both the request to invoke methods on the resource and the result is passed as *XML*. *DTD* and example of an invocation request can be found in Appendix C and the invocation result *XML* in Appendix D. Several methods can be invoked in one request to reduce the network traffic overhead.

12.5.4 Resource

As *AbstractResourceService* handles all communication and invocation on the server side, *Resource* handles everything on the client side. An instance of this class presents a client with the resource description and ways to invoke the declared methods on the resource. This is therefore the only class consumers of resources have to interact with. The generalization of resources makes it easy to implement workflows that utilize resources of great variety and enables clients to run methods on resources which they do not have, or want, the class definition of.

12.5.5 ResourceManagerService

For clients to utilize resources, they needs to be able to search for available resources. The *ResourceManagerService* is available through the *Framework* class.

The *ResourceManagerService* will scan the network for resources. When it finds one, it will ask for its resource description and add it to the list of known resources. The resource is then ready to be used by the client. It defines methods for iterating through and searching amongst the discovered resources. Any resource fetched from an instance of this class can be used directly, regardless of which client it is running on. It will remove any resource that has not responded to the scan for some time. All listeners are informed of findings and removals of resources from the list.

12.6 Workflow, Workflow Enactment and Scheduling packages

To test our arbitration service we need a workflow system. Within a workflow system lies the actual defenition of workflows, executing them and scheduling. The three packages, *workflow*, *workflowenactment*, and *scheduling* support these criterias.

12.6.1 Workflow package

This package defines the actual components of a workflow and the execution of statements. A workflow is built up of three main classes. The larges component is *ExecutableAcivity*. This is again divided in two, namely a *Goal* and an *Activity*. A *Goal* can consist of other goals and activities. An *Activity* can only have *Statements*. Both *ExecutableActivity* and *Statement* are extensions of the *Executable* interface.

Figure 12.7 shows the class diagram of the *Workflow package*.

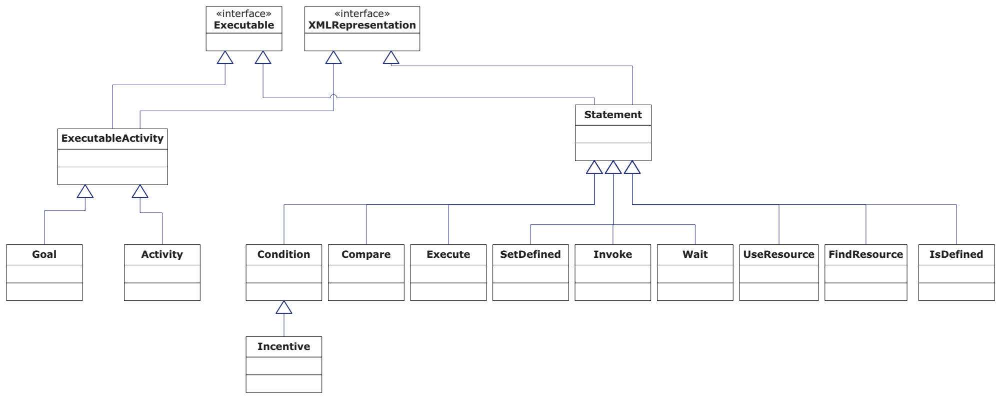
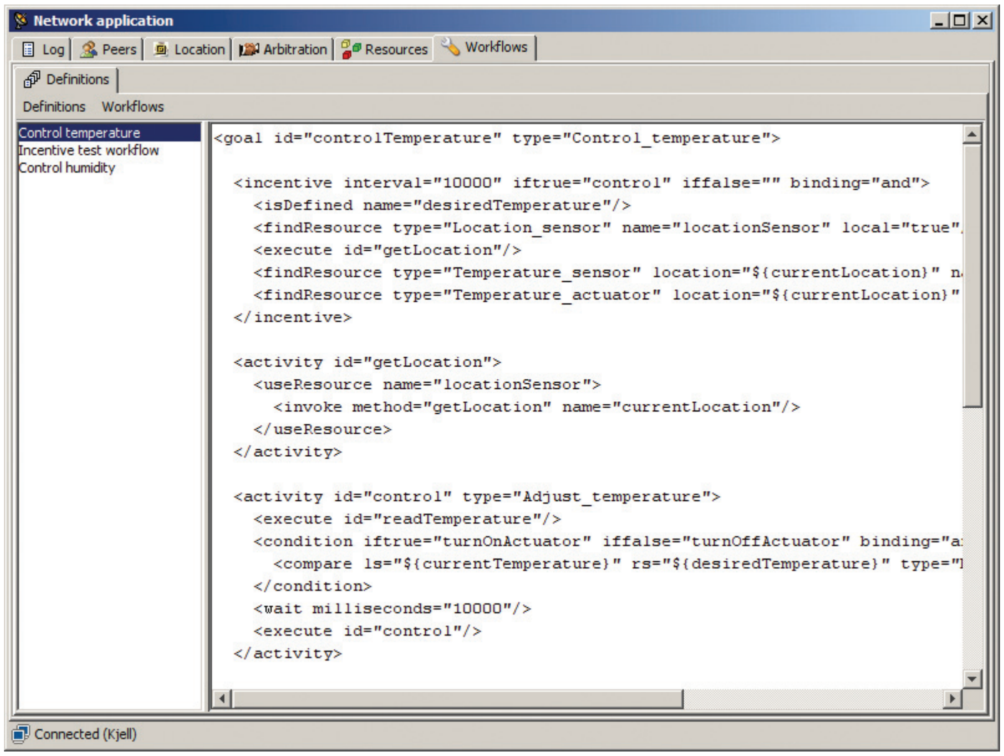


Figure 12.7: Shows the relations between the classes of the workflow package.

The *DTD* of the *XML* workflow definition is supplied in Appendix E and a complete example of a *XML* workflow definition, controlling the temperature of the room the client is in, is provided in Appendix F. The example is a workflow implementation of the scenario in Section 9.1.

Figure 12.8 shows a screenshot where workflows can be selected from the list of known workflows on the left or written as an XML definition directly into the textfield. The workflow can then be added to the schedule.



```
<goal id="controlTemperature" type="Control_temperature">

  <incentive interval="10000" iftrue="control" iffalse="" binding="and">
    <isDefined name="desiredTemperature"/>
    <findResource type="Location_sensor" name="locationSensor" local="true",
    <execute id="getLocation"/>
    <findResource type="Temperature_sensor" location="${currentLocation}" n
    <findResource type="Temperature_actuator" location="${currentLocation}"
  </incentive>

  <activity id="getLocation">
    <useResource name="locationSensor">
      <invoke method="getLocation" name="currentLocation"/>
    </useResource>
  </activity>

  <activity id="control" type="Adjust_temperature">
    <execute id="readTemperature"/>
    <condition iftrue="turnOnActuator" iffalse="turnOffActuator" binding="a
      <compare ls="${currentTemperature}" rs="${desiredTemperature}" type="
    </condition>
    <wait milliseconds="10000"/>
    <execute id="control"/>
  </activity>
```

Figure 12.8: Shows a screenshot of the tab where workflows can be selected or written in as an XML definition.

Executable

This is the interface that defines a workflow item that can be executed. As seen in Figure 12.7, it is the base for all workflow elements. It has two methods.

execute() : boolean is the methods invoked when the workflow item, a goal, activity or statement is to be executed. It returns a boolean value indicating if the execution itself was successful. If a fatal error occured during execution it will return *false*, and if no errors occured it will return *true*, regardless of any result of the execution.

The second method, *getResult() : boolean*, returns the result of the execution. This method is defined so that all classes extending *Executable* can be used in a conditional block.

Table 12.1 show the values returned from the different methods dependent on failures and result of execution for the *FindResource* statement. The *FindResource* statement looks up a resource, specified by parameters, from the *ResourceManager* and stores it in a named variable.

FindResource	execute()	getResult()
Execution error	*false*	*false*
Resource not found	*true*	*false*
Resource found	*true*	*true*

Table 12.1: Return value matrix for *FindResource*.

ExecutableActivity

This is the base class for the two blocks, *Goal* and *Activity*, of a workflow.

An *ExecutableActivity* has two important features. It has a parameter called *type*. This is a ontology individual that describes the nature of what the activity does. These ontologies can describe things like, *turn on heating oven*, *turn off lights*, and is the basis for arbitration on workflows covered later on.

The other important feature is the *id*. A specific id can be set on it, so that it can be executed from an *Execute* statement. All *ExecutableActivities* defined in a workflow *XML* will reside in a pool. The order of definition in the *XML* file is not important for the execution. This way the *ExecutableActivities* can call each other regardless of any predefined execution tree.

This functionality can be compared to *Procedural programming* where the workflow is considered as the class, *ExecutableActivities* as the procedures and *Statements* as each code line within a procedure. We saw this as a better approach that sequential workflows.

An *Activity* is an extension of *ExecutableActivity*. Within it, it can only hold *Statements*. These are executed in order of definition in the workflow *XML* file.

The method *execute()* will return true if all statements within an *Activity* were executed without any errors, and *getResult()* will true if all statements returned true on their *getResult()*.

The *Goal* class is the root of all workflows. As with *Activity*, it is an extension of *ExecutableActivity*. It defines a greater goal than an *Activity*, so where an activity might *turn on a heating oven*, a goal can *control the temperature* by combining several *Activities* or other *Goals*. A goal can therefore not contain any *Statements* directly but only other *ExecutableActivities*.

Since it is the root of a workflow, it must also define an entry point. This is done by the *Incentive* class.

Incentive is an extension of the statement *Condition*. In its definition, an *ExecutableActivity* can be specified to execute if the incentive is true and one to execute if it is false. In addition to this a execution interval is specified.

There are two things that separates it from *Condition*:

1. It is executed continually in the specified intervals regardless of any other *ExecutableActivities* it has started.

2. It triggers only on a change in the result. So when the result changes from false to true it executes the *ExecutableActivity* specified for true. When it changes from true to false, it stops the *ExecutableActivity* specified for true and starts the one specified for false.

The incentive should therefor check if all resources that is needed by the goal is present and available. To follow up on the workflow XML example in Appendix F. To control the temperature, it checks that a temperature sensor and a temperature actuator is present in the same room as the client. If so is, it will start to adjust the actuator according to the sensor. If the sensor suddenly disappears, it stops and executes the false *Executable-Activity*. Likewise, if the sensor appears again it will execute the true condition again, unless the false condition has halted the workflow with a control statement.

The goal also hold all variables for the executing activities and their statements. When a goal tries to resolve a variable and it does not have it, it will cascade upwards in the tree of goals to resolve it. This way one can combine many smaller goals into larger goals, sharing information.

All workflow elements are completely modifiable run-time. You can change any specified values, add, and remove statements, activities, and goals.

Statement

This class is the base for all single statements executed in a workflow. As described above, a statement is the smalest part of a workflow definition. It performs one and only one specific task. Several statements can be combined into one *Activity*. The currently defined statements are:

IsDefined Test whether a variable is defined or not. The name of the variable to check for is passed as an argument. If the variable exsists, regardless of value, *getResult()* returns true.

SetDefined Changes the value of a defined variable or creates the variable if not defined. Both the name of the variable and the value is passed as arguments.

Execute This statement executes a defined *ExecutableActivity* or performs a workflow control statement. If an *id* is supplied as an argument, it will look up the *ExecutableActivity* with the matching id and execute it. If a *control* argument is given, it will ask the workflow enactment service to execute a control command. This could be commands to halt the execution, pause it or control other workflows.

Condition This is a special form of statement as it is the only statement that can have sub-statements. Its task is to perform complex conditional blocks and can start execution of an *ExecutableActivity* for the result of either *true* or *false*. Using the argument *binding*, it can combine the *getResult()* with either *and* or *or*. When *and* binding is used, it will execute statements until finished or one statement returns false. When *or* binding is used, it will execute statements until one statement return true or finished. It will then execute the *ExecutableActivity* defined for respectivly true or false, dependant on the result.

Compare A single statement comparing to values. The two values can either be passed explicitly or as variables. It supports several methods. These are string equals, string equals ignoring case, boolean, numeric equals, numeric less, numeric less or equals, numeric greater, and numeric greater or equals. The result of the comparison is return with *getResult()*. Both *Condition* and *Compare* can be negated with an argument.

Wait Pauses the workflow for a given amount of milliseconds. This provided so the workflow can execute periodical tasks in a control loop. A control loop can be formed by calling the *Execute* statement to execute an *Activity* within the same *Activity*.

FindResource This looks up a resource from the *ResourceManager*. Parameters can be passed to search the *ResourceManager* for type, location or combinations of these. If a matching resource is found, it stores the resource in a variable named by an argument. A matrix describing the return values of the methods *execute()* and *getResult()* can be found in Table 12.1.

UseResource This is a wrapper for the *Invoke* statement. It is provided so that many *Invokes* can be made on a resource in one statement, saving network resources. This statement has only one argument with the variable name holding the resource to use.

Invoke One ore more of this statement can be defined within a *UseResource* statement. It will invokes a method on the resource fetched by *UseResource*, with given parameters. The result of the invocation will be placed in a variable named by an argument.

All arguments or parameters to statements can either be entered explicitly or looked up in the workflows set of variables. The syntax for an explicitly passed value is the value itself. If the value is to be found in a variale, the variable name is surronded by ${variable-name}.

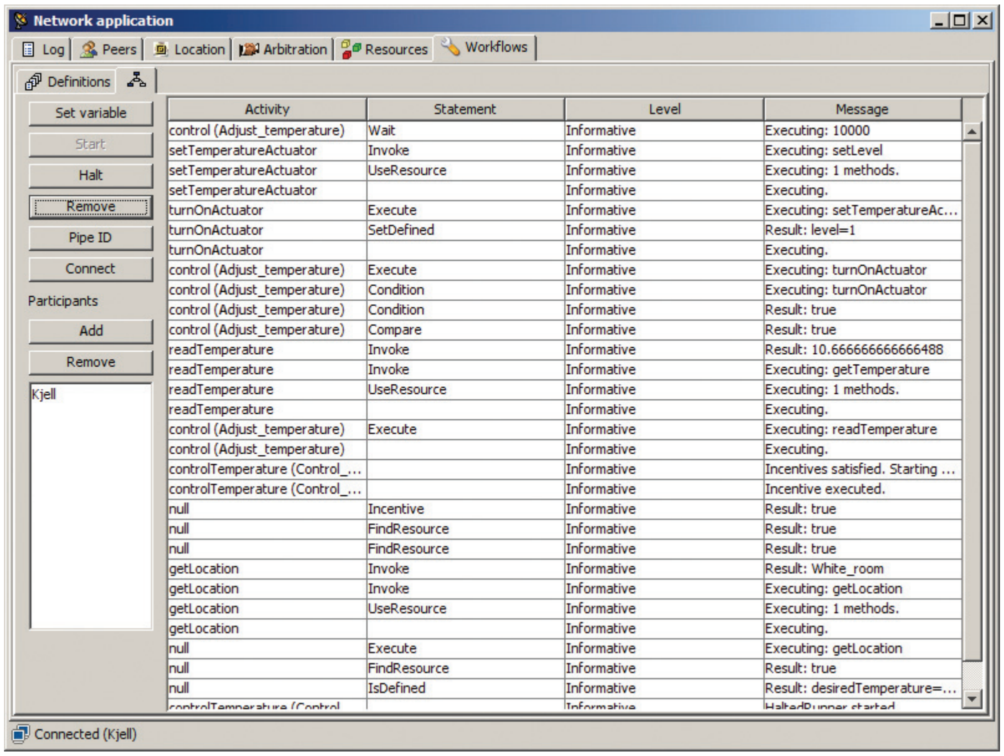
`<wait milliseconds="5000"/>` results in a 5 seconds sleep, while
`<wait milliseconds="${sVar}"/>` result in a period of sleep in milliseconds equaling the value found in the variable *sVar*.

For more detailed information on the workings and possible arguments, we refer to the implementation JavaDoc or the DTD specification in Appendix E. The JavaDoc is not provided with this report as it is not considered important for answering our research questions.

12.6.2 Workflow enactment package

This package has the responsibility of executing the actual workflows and monitor workflows executed on other peers. It has four major classes. *Workflow*, *WorkflowContext*, *RemoteWorkflowServer*, and *RemoteWorkflowConnector*. The two latter are for cooperating with other peers.

Figure 12.9 shows a screenshot of the workflow from Appendix F executing on a peer. There are commands for defining variables in the running workflow context, commands to start and stop the workflow, connect to remote workflows, and manage participants. The log shows the execution progression for that workflow.



Activity	Statement	Level	Message
control (Adjust_temperature)	Wait	Informative	Executing: 10000
setTemperatureActuator	Invoke	Informative	Executing: setLevel
setTemperatureActuator	UseResource	Informative	Executing: 1 methods.
setTemperatureActuator		Informative	Executing.
turnOnActuator	Execute	Informative	Executing: setTemperatureAc...
turnOnActuator	SetDefined	Informative	Result: level=1
turnOnActuator		Informative	Executing.
control (Adjust_temperature)	Execute	Informative	Executing: turnOnActuator
control (Adjust_temperature)	Condition	Informative	Executing: turnOnActuator
control (Adjust_temperature)	Condition	Informative	Result: true
control (Adjust_temperature)	Compare	Informative	Result: true
readTemperature	Invoke	Informative	Result: 10.666666666666488
readTemperature	Invoke	Informative	Executing: getTemperature
readTemperature	UseResource	Informative	Executing: 1 methods.
readTemperature		Informative	Executing.
control (Adjust_temperature)	Execute	Informative	Executing: readTemperature
control (Adjust_temperature)		Informative	Executing.
controlTemperature (Control_...		Informative	Incentives satisfied. Starting ...
controlTemperature (Control_...		Informative	Incentive executed.
null	Incentive	Informative	Result: true
null	FindResource	Informative	Result: true
null	FindResource	Informative	Result: true
getLocation	Invoke	Informative	Result: White_room
getLocation	Invoke	Informative	Executing: getLocation
getLocation	UseResource	Informative	Executing: 1 methods.
getLocation		Informative	Executing.
null	Execute	Informative	Executing: getLocation
null	FindResource	Informative	Result: true
null	IsDefined	Informative	Result: desiredTemperature=...
controlTemperature (Control		Informative	HaltedRunner started

Figure 12.9: This screenshot shows the workflow from Appendix F executing on a peer.

Workflow

This is a wrapping class for workflows and the only workflow interface towards the application. It creates a *WorkflowContext* for the workflow to execute within and a *RemoteWorkflowServer* for other peers to connect to and monitor that workflow. It will also handle the list of participants that are allowed to connect and be a part of the workflow.

WorkflowContext

This is the running context of the workflow. It will receive any control statements from the executing workflow and user interface. It is this context the workflow execution checks for the *halt* flag when performing a controlled and clean exit. It will also give the workflow access to a log. In this way, it can either pass the log entries on to the application log or keep a separate log for each executing workflow. The last alternative is the one implemented in our application.

RemoteWorkflowServer

This is a class that enables other workflows to connect to the workflow this server is associated with. Upon workflow instanciation, it creates a JXTA server pipe. It will only respond to peers that are added as participants to the workflow.

If a peer requests, it will respond with a status message containing the state of execution, e.g. if it is running, and the remaining log entries recieved since last time it sent a status message to that peer. It maintains a counter that is incremented each time a log message is added. The current counter value is passed with the status message. When a peer requests a status message it will provide the *RemoteWorkflowServer* with the value of the counter from the last call. This way it can calculate and send only the new entries.

The server does not support recieving workflow control commands or workflow adaption messages, although this is supported by the implementation.

The DTD and an example of the status message sent from this server to participants is provided in Appendix G.

RemoteWorkflowConnector

An instance of this class is created in a *Workflow* when a request has been made to connect to a workflow executed remotely. It will stop the current execution of the local workflow, connect to the remote, and periodically ask for status messages. The status messages include the log messages added to the remote workflow since the last time a status message was recieved.

If *RemoteWorkflowConnector* does noe recieve a response for a configurable amount of iterations, it will asume that contact is lost with the remote workflow or the peer in general. It will then stop monitoring and return control to the local workflow definition.

This is a strategy used in arbitration for some of the goals in need of cooperation.

12.6.3 Scheduling package

This package holds only one important class; *Scheduler*. Currently, this package only supports a very basic scheduler.

Scheduler

An instance of this class holds a list of currently loaded workflows in the application. It is accessible through the *Framework* class as one of the major components. This is the only interface for the application to manage workflows.

As with several other classes, it will also create a JXTA server pipe. It is used to give other peers access to the description of all currently running local workflows. Primarily this function is used by the arbitration service in our implementation, but it can also be used so that other can see what workflows are running on you system and thereby create an awareness amongst cooperating entities.

The description only holds the ontology individual for type, ontology individual for location, and address information of the workflow for those that have this information supplied. See Section 12.7.2 on *ConflictDiscoverer* for a more detailed description of this XML message. The DTD and an example can be found in Appendix I.

12.7 Arbitration package

This is the package that that searches for conflicts and performs arbitration on workflows. The *cbr* sub-package has all classes defining the CBR implementation of our application.

Figure 12.10 is an activity diagram and Figure 12.11 is a message sequence diagram of the search for possible conflicts and the arbitration process.

12.7.1 Arbitration

This class instantiates and controls everything needed to perform arbitration internally and with other clients. It is the only class needed by other components to perform arbitration and is accessible thourgh the *Framework* class.

It will instantiate the *ArbitrationCBR* class, a *ConflictDiscoverer* and a private class controlling a JXTA server pipe for communication with arbitration services on other clients.

ConflictDiscoverer is responsible for discovering possible conflicts and fires an event if so happens. This class is explained later on.

When a conflict is identified, this class performs arbitration on the workflows. It currently supports three strategies. For all three there is two variants, one to perform the strategy on the local client and one to perform it on the remote client. This support is useful as it is dependant on the order of the conflicting workflows. In a conflict between two different workflows, there might always be desirable to let a specific of them wait. This way it can ensure that one type of workflow is halted or put on wait when it comes into conflict with a specific other. The tree strategies are:

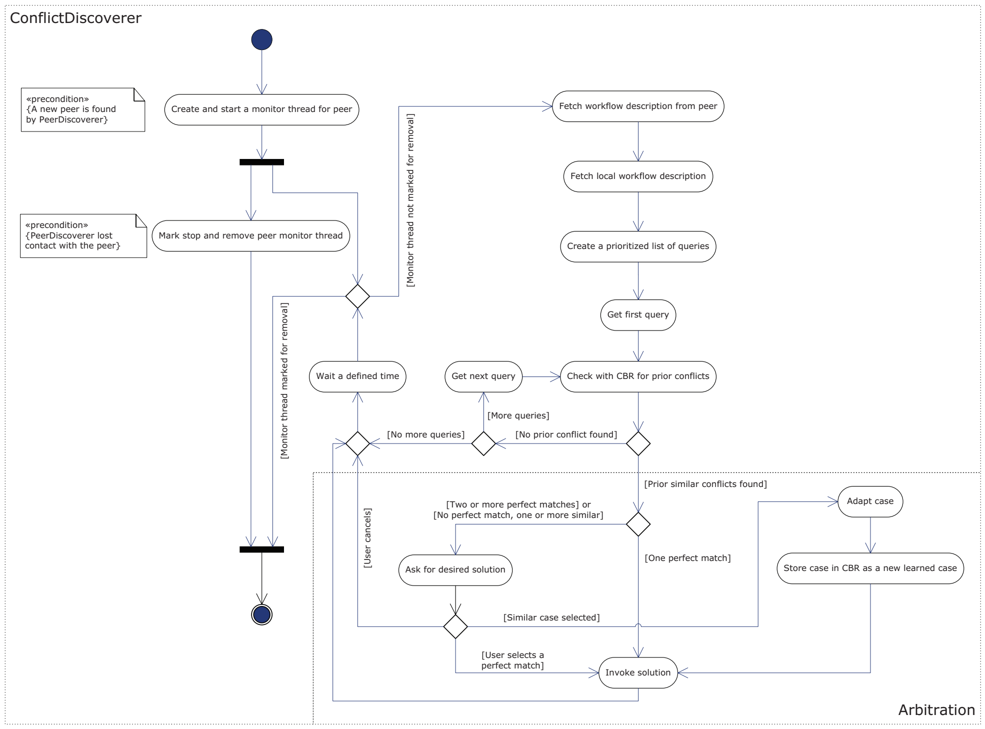

ConflictDiscoverer
«precondition» {A new peer is found by PeerDiscoverer}
Create and start a monitor thread for peer
«precondition» {PeerDiscoverer lost contact with the peer}
Mark stop and remove peer monitor thread
[Monitor thread marked for removal]
[Monitor thread not marked for removal]
Fetch workflow description from peer
Fetch local workflow description
Create a prioritized list of queries
Get first query
Wait a defined time
Get next query
Check with CBR for prior conflicts
[More queries]
[No more queries]
[No prior conflict found]
[Prior similar conflicts found]
[Two or more perfect matches] or [No perfect match, one or more similar]
Adapt case
[User cancels]
Ask for desired solution
[One perfect match]
Store case in CBR as a new learned case
[Similar case selected]
[User selects a perfect match]
Invoke solution
Arbitration

Figure 12.10: Shows the process of searching, adapting, arbitration, and learning cases.

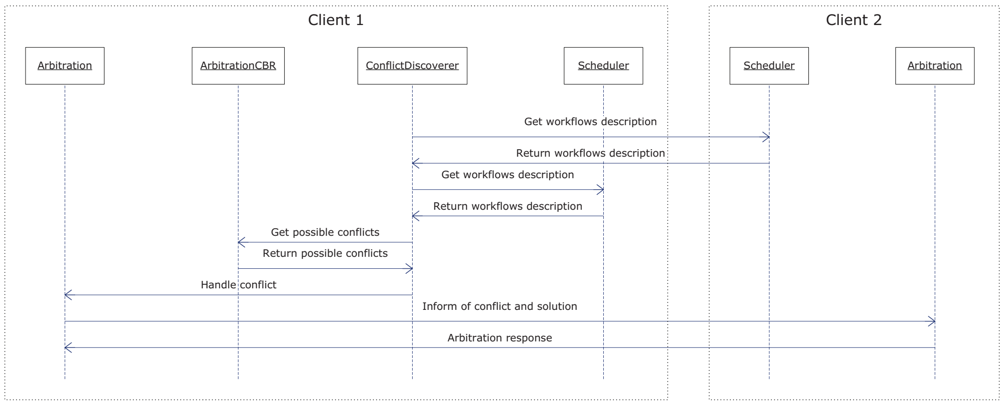

Client 1
Client 2
Arbitration
ArbitrationCBR
ConflictDiscoverer
Scheduler
Scheduler
Arbitration
Get workflows description
Return workflows description
Get workflows description
Return workflows description
Get possible conflicts
Return possible conflicts
Handle conflict
Inform of conflict and solution
Arbitration response

Figure 12.11: Shows the messages passed between different components in the arbitration process.

Halt This strategy halts a running workflow based on a conflict with another. The workflow is completly stopped and must be started manually to execute again. Which workflow is stopped, is dependant on the solution description. It will not be removed from the application scheduler as it might still be needed.

Wait A workflow waits for another workflow to finish before it continues. This is done by pausing the workflow in conflict and creating a local workflow that connects to the remote by using the *RemoteWorkflowConnection* covered earlier in this document. When *RemoteWorkflowConnection* either looses contact or the remote workflow is halted or finished, it will return control to the waiting workflow causing it to continue execution. The generated workflow for remote connection is then removed from the scheduler.

Adapt Adaption is a complex task to perform on two workflows. In this proof-of-concept implementation we have only made one adaption strategy. It creates a workflow that combines the two workflows in conflict. This only supports workflows that are of the same nature (tries to achieve the same goal). It does so by gathering all numeric variables in the executing workflows' context and for every variable name that is common to both workflows it creates a mean value. These new variables are then the basis for the merged workflow. The original workflows are then put on wait while the cooperation is in progress. When the cooperation is disbanded the original workflow is reactivated.

Figure 12.12 shows a screenshot of an adapted solution to the temperature conflict described in Section 9.1. There are two workflow tabs. The first is the original workflow controlling temperature and the second, the one visible, is the adapted solution. As can be seen in line 1 in the log, the desired temperature is 22℃. This is a mean between the two clients that had 20℃ and 24℃ as their respective desired temperatures.

The *Arbitration* class communicates with arbitration services on other clients. The DTD specifying the *XML* message and an example of the *adapt* strategy above, is provided in Appendix H.

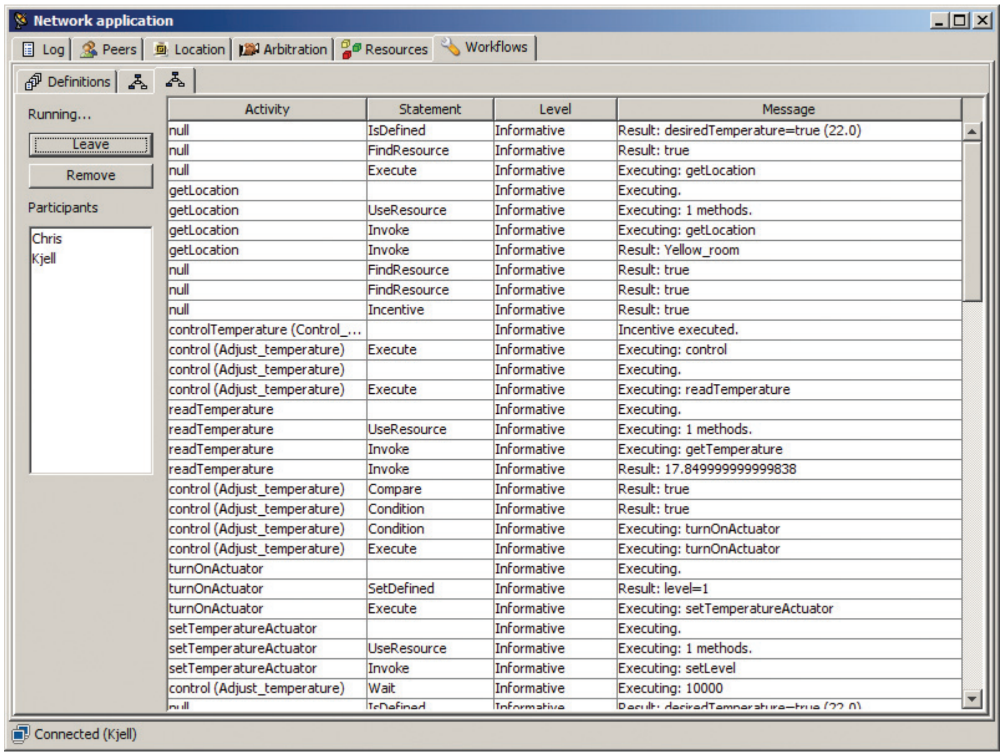
A possible conflict may not always be a perfect match. If a working case resembles a known conflict to a certain extent, it will be regarded as a conflict. This is made possible by the ontology classifications to enable the system to learn. If a solution is chosen, it is adapted by copying the classifications of the working cases and stored. This way the system learns new conflicts from old.

Whenever an arbitration service performs arbitration, it sends the case that it based the solution on, possibly adapted, to the other client. This is a way of ensuring that all clients are aware of the basis of the arbitration decicion and so they can learn the case, if not already known. This way, knowledge of conflicts is shared between clients. The case is passed as a part of the arbitration message defined in Appendix H.

12.7.2 ArbitrationCBR

This is the class controlling the jColibri system. Here the different tasks and methods of the four CBR-cycles are defined, configured and executed. After instanciating the jColibri core, *ArbitrationCBR* loads the different configuration packages.

All task and methods are configured during instanciation. Task and methods are roughly defined as in Figure 5.8, where the vertically arranged boxes are tasks and the decompo-



Activity	Statement	Level	Message
null	IsDefined	Informative	Result: desiredTemperature=true (22.0)
null	FindResource	Informative	Result: true
null	Execute	Informative	Executing: getLocation
getLocation		Informative	Executing.
getLocation	UseResource	Informative	Executing: 1 methods.
getLocation	Invoke	Informative	Executing: getLocation
getLocation	Invoke	Informative	Result: Yellow_room
null	FindResource	Informative	Result: true
null	FindResource	Informative	Result: true
null	Incentive	Informative	Result: true
controlTemperature (Control_...		Informative	Incentive executed.
control (Adjust_temperature)	Execute	Informative	Executing: control
control (Adjust_temperature)		Informative	Executing.
control (Adjust_temperature)	Execute	Informative	Executing: readTemperature
readTemperature		Informative	Executing.
readTemperature	UseResource	Informative	Executing: 1 methods.
readTemperature	Invoke	Informative	Executing: getTemperature
readTemperature	Invoke	Informative	Result: 17.849999999999838
control (Adjust_temperature)	Compare	Informative	Result: true
control (Adjust_temperature)	Condition	Informative	Result: true
control (Adjust_temperature)	Condition	Informative	Executing: turnOnActuator
control (Adjust_temperature)	Execute	Informative	Executing: turnOnActuator
turnOnActuator		Informative	Executing.
turnOnActuator	SetDefined	Informative	Result: level=1
turnOnActuator	Execute	Informative	Executing: setTemperatureActuator
setTemperatureActuator		Informative	Executing.
setTemperatureActuator	UseResource	Informative	Executing: 1 methods.
setTemperatureActuator	Invoke	Informative	Executing: setLevel
control (Adjust_temperature)	Wait	Informative	Executing: 10000
null	IsDefined	Informative	Result: desiredTemperature=true (22.0)

Figure 12.12: Shows an adapted workflow based on two workflows trying to control the temperature of a room.

sition of those are methods. Each task and methods is customizable. Which class is to be responsible for each task or method is decided by a complete class name. jColibri requires that all classes used in tasks or methods are described in their configuration files. When each method is configured and each task is configured with methods, jColibri is ready to be used.

jColibri comes with Jena. This is a Java framework using RDF models, created by IBM, and is the base for our use of ontologies. When *ArbitrationCBR* loads jColibri it, in turn, loads Jena. The loaded instance of Jena is then, through a jColibri bridge made available to the whole implementation through the *Framework* class.

jColibri is implemented with one context that the cases resides within. Case searching, adaption and learning can only be done within this context. Since our system is based on many threads that may access *ArbibrationCBR*, and thereby jColibri, at the same time, we needed to make the implementation thread safe. This is done by defining synchronized methods for searching possible conflicts and to store learned cases. An activity diagram showing this behaviour can be found in Figure 12.13.

To get a working implementation, some jColibri tasks and methods had to be specialized to work.

Connector

The connector is the component reading and writing the case base to and from persistent storage. Because of problems with jColibri reading in from file and instantiating the ontology individuals from Jena, we had to implement our own connector. More about this problem is described in Technology evaluation in Section 16.3 in the discussion chapter.

CaseSimilarity

This class computes the similarity between two cases. It is done by calculating the distance between the two cases' describing individuals in the ontology tree. This way it determines how close the nature of operation of the cases are and can determine if there is a possible between them. If they are close, there might be a conflict and *Arbitration* can perform adaption of the case. It returns the percentage of nearness in the interval $[0, 1]$.

If an individual is a perfect match, it returns *1.0*. If it is part of the same leaf it returns *0.75* and if it is one leaf up or down in the tree, it returns *0.5*. For any other cases it return *0*. An illustration of this can be seen in Figure 12.14.

Each case has four parameters. They are:

Case1 This is the ontology individual of the first workflows entity.

Case2 This is the ontology individual of the second workflow entity.

Location This is a double describing the importance of location nearness.

Participants Unused in this implementation, but intended to inform about the importance of the composition of participants executing the workflows.

The total case similarity between cases is calculated by jColibri as the mean value of the similarity of these four properties. jColibri does not have the built in function of properties

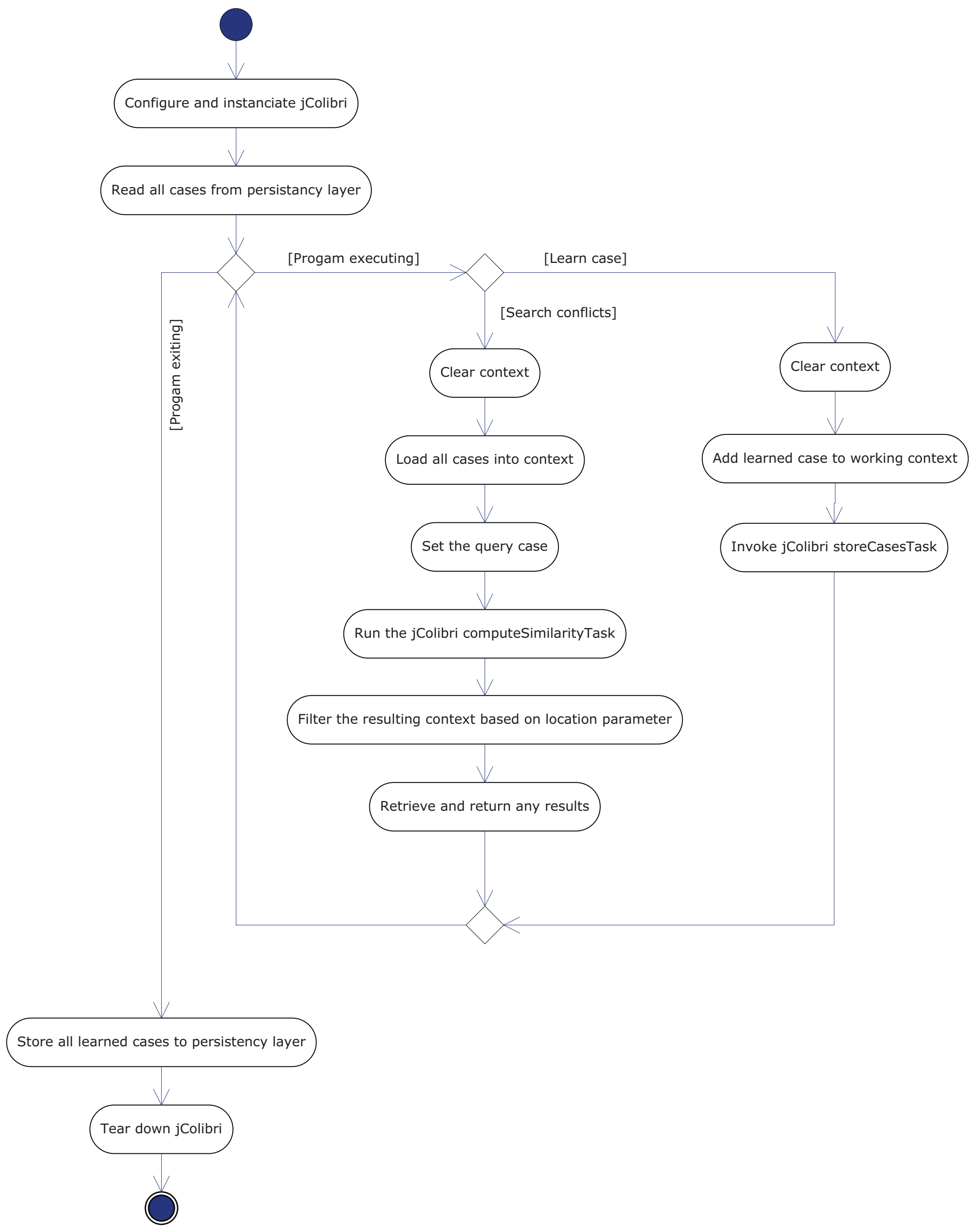

Configure and instanciate jColibri
Read all cases from persistancy layer
[Progam executing]
[Learn case]
[Search conflicts]
[Progam exiting]
Clear context
Load all cases into context
Set the query case
Run the jColibri computeSimilarityTask
Filter the resulting context based on location parameter
Retrieve and return any results
Clear context
Add learned case to working context
Invoke jColibri storeCasesTask
Store all learned cases to persistency layer
Tear down jColibri

Figure 12.13: Shows the process of searching and learning cases in *ArbitrationCBR*.

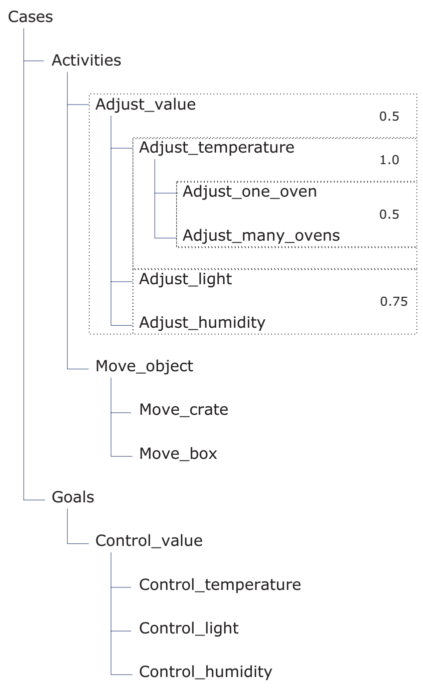


Figure 12.14: Shows how the similarity between ontology individuals is calculated.

being able to be exclusive. During the calculation of similarity, location and participants always return *1.0*. A filtering based on location is done after jColibri has calculated the similarity. We call this way of filtering *context triggers* and more about this strategy can be read in Section 16.3 in the discussion chapter.

The *location* parameter of the query-case is calculated the same way as the case ontologies. If the value of *location* is 1, it means that there only is a conflict if the two workflows are executed in the same location. If it is 0, it means that location is of no importance. If the result of the calculatiion of the distance between the locations of the two workflows is higher than the value stored for *location* in the case then tells us if we have a conflict.

In addition to the four parameters describing the conflict, it has two parameters describing the solution. These are *Solution* and *Parameter*. *Solution* is an ontology individual describing the solution strategy and *Parameter* is string intended to provide important information on how to invoke the solution strategy.

ConflictDiscoverer

This class searches for possible conflicts. Figure 12.10 shows the activities of the *Conflict-Discoverer*.

It will listen to the *PeerDiscovery* service and maintain a list of conflict searching threads, one thread per known peer. Periodically, each thread will ask the peer it is responsible for, to supply a description of the workflows it is executing. It will then create a prioritised list of cases which will be matched up against the known conflicts by *ArbitrationCBR*.

The list is created based on workflow description *XML* messages. These XML workflow description messages are fetched from the *Scheduler* of the peer each searching tread is responsible for and is matched up against the local workflow. This list is created by first matching every node in the same position in the tree. Then it completes the list by adding all combinations not yet in the list. Figure 12.15 shows an illustration of this process.

The DTD and an example of such a workflow description *XML* message can be found in Appendix I.

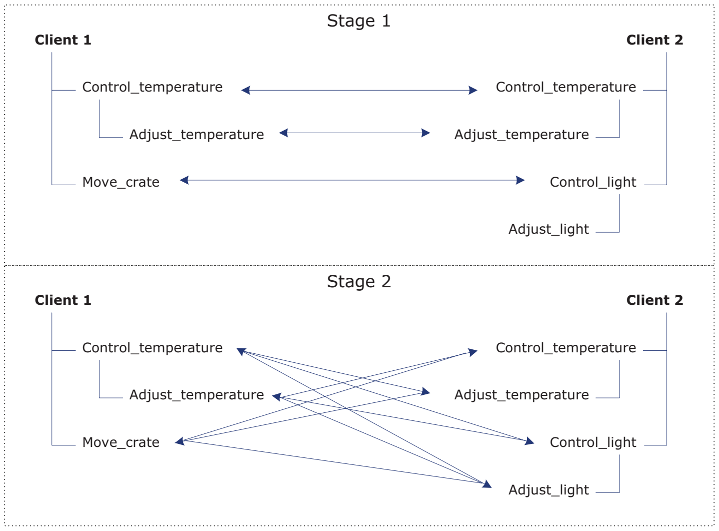


Figure 12.15: Show how the conflict discoverer sets up the list of conflicts to search for.

PART IV

Planning

CHAPTER 13

Scenario - Dice factory

Motivation

Planning can be very important in connection with performing work processes. Plans can represent the difference between success and failure in achieving the respective goals of a process. However, as the following scenario shows, work processes are dynamic constructs which might demand restructuring in order to achieve best possible results. Plans must therefore have the ability to adapt to changes in the environment.

Scenario

CuttingWood is a company that makes different type of customized wood products. One day they get an order to make 5000 uncoated dices to be delivered 2 days later. The next day they get an order for 3000 red coated dices for another company, with the same delivery date as the first order.

Their automated production system is able to produce 4000 dices a day, and calculates when to start producing the orders. It has already started producing the first order and has finished 1000 dices when the second order comes in. It checks the production orders and sees that the coating for dices needs one day to finish. If it continues to produce the first order it would not be able to deliver the second order in time.

But the automated production system is smarter than that. It figures out that it can halt the first production and start producing the second order, which it would finish during the day. Then these dices can be sent off to coating, and the system can continue production of the first order. Doing this, it can actually finish both orders in time.

The system realizes that the first order has priority since it is not only is placed first, but also is currently in production and has a greater potential for profit due to its size. By checking the time table and availability of the production system, it realizes that coating and production of dices could be done in parallel. Acknowledging this, it de-prioritizes the first order to make way for the second one, so that it can be shipped off to coating whilst the system finishes producing the first order.

The production system then tells the ordering system to accept order number two, and invokes its re-planned work processes. In this way ordering of tasks at CuttingWood is done in a most efficient way, always utilizing the full capacity of the available production equipment.

Challenges in scenario

- Resolving which processes can be executed in parallel
- Can the work process be satisfied if reordered
- Knowing the availability of resources and the interdependency between them
- Guaranteeing delivery based on uptime and throughput of the production system

CHAPTER 14

Workflow generation

When planning how to perform work, it can be wise to divide the task into its constituent parts (activities). Activities would then perhaps have to be performed in a certain order to achieve the goal of the task. These chains of activities are represented as workflows (Section 5.5.4 and Chapter 13). Activities may need to have certain conditions fulfilled before they can execute. As this foundation (context) may change and alter the situation, workflows should be dynamic (adaptive workflows). By this we mean that they should be able to rearrange activities or import substitute activities which are suited to adapt to the new situation. Workflows then make up the foundation for creating a plan, which also must be dynamic in the sense of available resources and context states.

In this Chapter, we start with a proposal on how to fetch context, as context is a foundatoional part of situated planning (Section 5.5.2). After this, we elaborate how to possibly generate workflows based on current context with minimal human interference and dynamically keep these up to date as the execution context changes. We then look at how activities' effect on context can be stored, used, and improved.

14.1 Gathering context - the context service

The *Smart work processes* (SWP) architecture, the *Java Context Aware Framework* (JCAF), and the *Context Toolkit* (Section 6.2) all provide versions of a component that obtains context from the environment, abstracts it, and provides it to subscripting components or services.

We want to keep the property of personalised representation of context, as well as processing all kinds of context in one service. Therefore, we have moved away from the *Context widget* of the *Context Toolkit* and instead focused on merging the *Context services* provided in SWA and JCAF (Figure 14.1).

In this figure we show what functionality we think such a service should offer. The setting is represented by a *Smart space* (Section 3.3), where information resources reside along with context demanding services. The *Context service* will bridge the gap of this *supply*

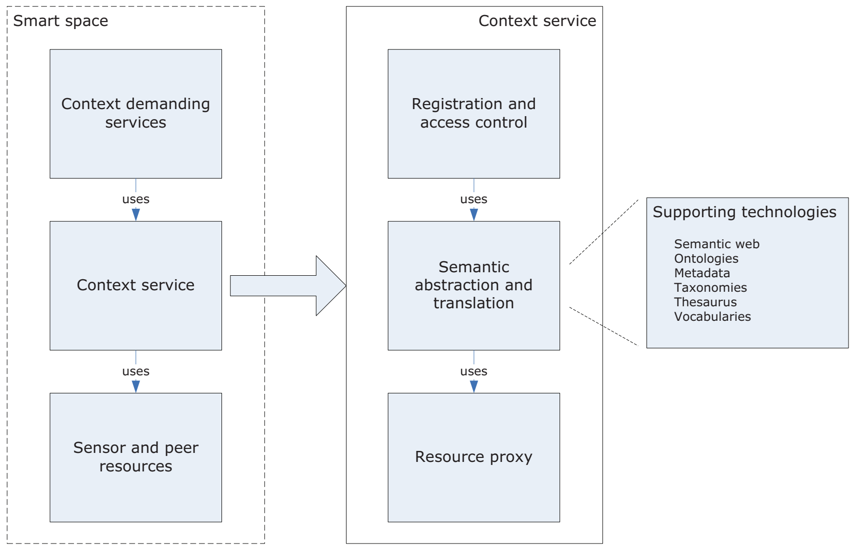


Figure 14.1: How we imagine a *Context service* (based on a fusion of the existing *Context services* provided by Sørensen et al. [88, 89], and Bardram [14]).

and demand situation, making information from heterogeneous resources look the same to any subscriber.

The *Context service itself* consists of a *Registration and access control* which controls the subscribers and their requests for context information regarding both type and semantic abstraction level. The *Semantic abstraction and translation* component provides the correct semantic representation of the requested context type to each subscriber, using, e.g., *Semantic web*, *ontologies*, *metadata*, *taxonomies*, *thesaurus*, and *vocabularies*, as mentioned in section 4.1.2.

A smart space can have a dedicated resource proxy to reduce network traffic, control and schedule resource use, and perhaps control access rights. With a stationary proxy residing in the *Smart space*, the *resource proxy* component in the client only provides an interface for this. If a stationary proxy is non-existing, the client component can act as a resource proxy to serve other clients when present in the *Smart space*.

We have not elaborated if and how to use historical context data for reasoning purposes, but on an entity with enough memory and processing capacity this can be added to the *Context service* model. If present, such a component should also contain or manage a history of semantic abstraction lookups already performed, so that these could be ”learned” and a local lookup could be adequate.

With the gathering of context in place, we can start elaborating how to create workflows based on context and other constraints.

14.2 Obtaining activities from goals - generating workflows

In order to automate as much as possible regarding the creation of workflows, the constituent parts of such workflows must be able to be localised based on the simple knowledge of what we want to achieve, or how to reach the goal of workflows. In the following we describe the structure of activity descriptions and what technologies that can be used to serve this purpose.

- If not available, provide a manual description of each activity needed to achieve the goal. This description should contain constraints (when can the task be executed) and success criteria (when are the task complete). Some tasks may also include quality criteria (preferred fidelity). Dependencies in the goal–activity hierarchy are only downwards, which indicates that a goal can obtain knowledge of what activities that is needed to achieve it, but that an activity does not know whether it belongs to a certain goal or not. An activity is then a reusable asset, but to exploit this reusability we must be able to determine the appropriateness of the activity regarding the goal.

- As activities are described, they are also saved and made available in the network. Because they are provided by different users with different intents, activities are not necessarily inter-exchangeable without further ado. Different users may define different activities for achieving the same goal or defining similar activities to achieve a different goal. User intent could be conserved by using specific tasks to achieve a specific goal. To make activities universally traceable, they could be given a unique address as part of the activity scheme. This could be done by using, e.g., the *resource description framework* (Section 5.3.2) provided by W3C.

- To gain best possible results when using activities provided by others, we suggest that *reinforcement learning* (Section 5.4.5) is used to study the results different activities provide (due to e.g. fidelity and other criteria) and that the results from this process is may be included in the activity description or provided by other information sources. In this way we can map what combinations provide the best result in achieving the goal.

- In order to automate as much as possible, we want to reuse and modify activities used in similar situations. To gain an overview of different activities' relationships, we suggest that they are described by *ontologies* (Section 5.3.2). Ontologies offers a description of the concept (activity), the activity's properties, and the restrictions of these properties. This implies a schema structure which is possible to combine with the activity properties already described.

- For a goal to obtain knowledge of what activities it need to execute, we suggest that *case-based reasoning* (Section 5.4.1) is used to recognise what has been good solutions in the past.

When the necessary and optimised activities to achieve a goal (based on current context) are obtained, these constitute the initial workflow. As the context state changes, e.g., based on other actors' plan executions, we have to adapt to these changes.

14.3 Adaptive workflow

Even though we base our planning aproach on workflows, coordination amongst activities needs to be performed between work processes and the work environment. We must perform context-based reasoning to reach process goals, and perform actuations which may change the workflow. This, amongst others, is supported by the SWP architecture (Section 6.1), which can be used to facilitate the coordination of multiple actors. The implementation of such mechanisms are usually addressed by using multi-agent systems (Section 5.2.3).

Managing the dynamics of ad-hoc activities and process changes implies that we must solve conflicts amongst actors. To support the dynamics, we must solve these conflicts in an automated manner, requiring minimal intervention from users. As an alternative to agents, we suggest the already elaborated and implemented *Arbitration service* (Part III) for this purpose. A solved conflict will most likely apply changes to a plan, triggering readaptation and rescheduling.

An adaptation can result in restructuring of the workflow. Methods for supporting this includes: *late binding*, *on-the-fly workflow process composition*, *partial execution*, *reusable process fragments and component libraries*, *selection constraints*, *termination constraints*, *build constraints*, and *workflow templates.* These methods are explained in [48], and will not be elaborated further.

How workflows are adapted to the new situation may also change. Context states could trigger adaptation of workflows both by changing the existing activities or introducing new activities. However, we keep to the latter, as this best makes use of the following approach.

Using *expert systems* (Section 5.4.2) for the adaptation of workflow require that rules are made for all different exceptions. To make such a system more robust, we could add the properties of *fuzzy logic* (Section 5.4.4). This would lead to a more tolerant workflow, not failing its plan because of some unexpected variants in the context. *FuzzyJess* [21] is an example of a system that combines a rule-based expert system and fuzzy logic.

We will however use another approach for adaptation, namely *Case-based reasoning* (Section 5.4.1). CBR offers a knowledge base of experience from previous cases. If the contextual foundation changes so an activity in a workflow, and thereby the workflow itself, cannot execute, a different set of activities which satisfies the new conditions can be automatically obtained and selected. However, this is only possible if the *case-base* contains the needed activities from previous executions. If not, the alternatives are to adapt the existing activities to the new situation or manually enter a new solution. As more variants of activities are stored in the *case-base*, the system will grow more robust and less dependent of human interference.

In Figure 14.2, we have extended the workflow definition presented in Section 5.5.4 to contain a conceptual CBR solution.

In the figure, when planning is done, tasks are ready to be executed. This is done using the resources available in the actors environment. Feedback on each single task (activity) execution and result, accompanied by feedback on how the goal was fulfilled, is the subject for reasoning about how planning of work could be improved in the future. Now that we have historical data, these may be used at a later occasion when a similar goal or task is to be executed by any entity with access to the case-base.

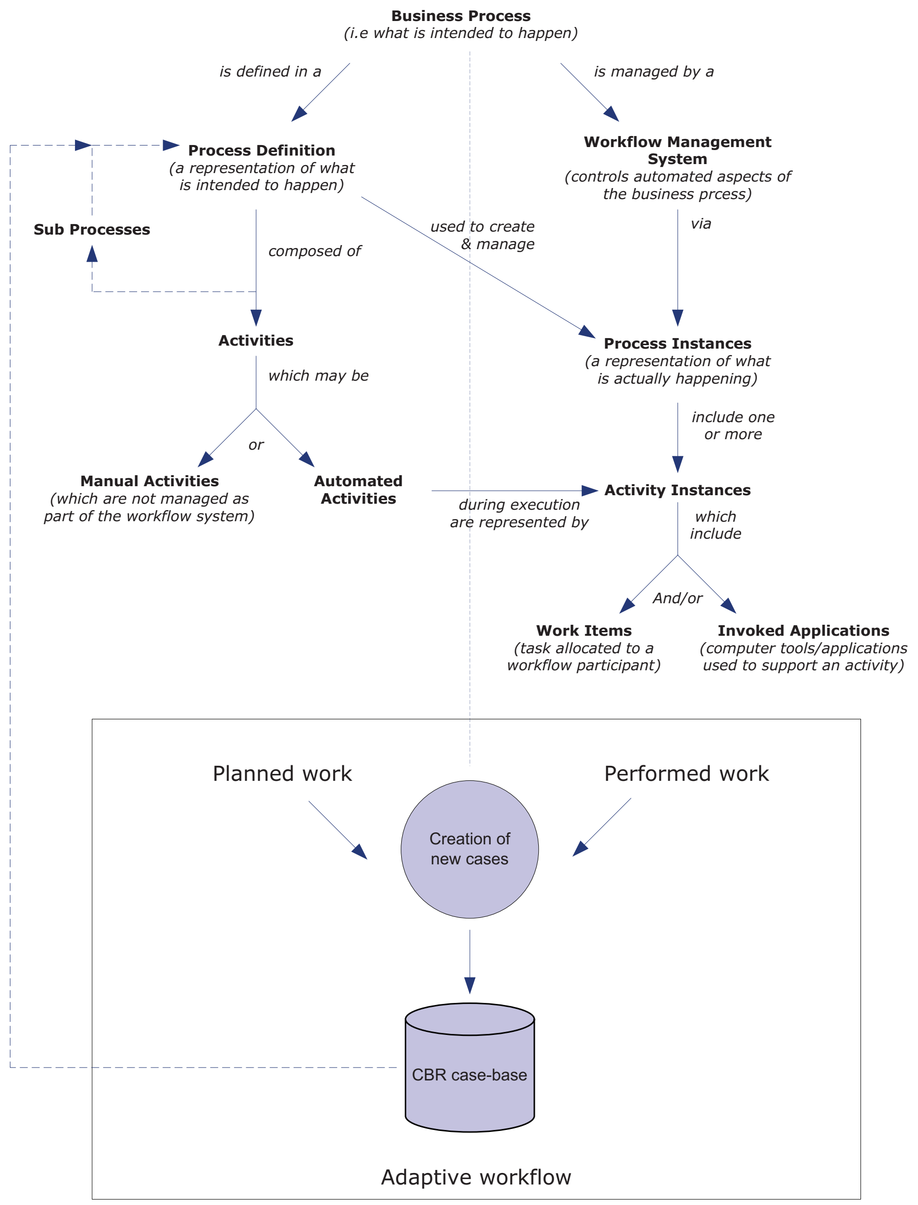

Business Process
(i.e what is intended to happen)
is defined in a
is managed by a
Process Definition
(a representation of what is intended to happen)
Workflow Management System
(controls automated aspects of the business prcess)
Sub Processes
composed of
used to create & manage
via
Activities
Process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

Figure 14.2: Adaptive workflow by using CBR (based on [28]).

An activity may be affected by context or context changes in several ways. When activities are dependant on the contextual state, they need to include specifications and rules on how context relates to the activity, hence how context affects the activity and how the activity affects context. The first can be obtained by having constraints and success criteria in the activity description and/or central databases, as described abowe. The latter is perhaps more complicated, as an activity's effect on context is not necessarily deterministic.

One of the important information types related to an activity, is then how its execution is expected to affect context. This information is valuable when trying to construct a future context and thereby construct better plans.

14.4 Obtaining, storing, and improving activities' context influence

Every activity added or changed in a workflow will affect the context. How this information is obtained can be solved in several different ways:

- The information could be part of the activity's scheme.
- It could be collected from an external database, with a reference to one or more activities.
- It could be fetched through knowledge systems.

These methods are very similar to how we picture the gathering of already defined activities in order to achieve a goal. How the information is actually resolved, depends on its complexity and size, which we do not elaborate in this thesis. The different methods have their pros and cons, such as different portability abilities and resource demands.

However, to improve the knowledge of how activities affect the environment, we see *neural nets* (Section 5.4.6) and *reinforcement learning* (Section 5.4.5) as possible candidates when learning of the cause and effects of activity execution. A problem with this approach is the need of execution in a isolated environment, without other actors influencing the results by their own executions.

Now that gathering of current contextual information, making workflows, workflow adaption, and obtaining activities context influence are elaborated, the next thing to do is to create an actual work plan based on the best combinations of workflows that fulfils the contextual requirements.

CHAPTER 15

Scheduling service

In our depth study [75], we developed a service architecture for planning and scheduling activities in a dynamic environment. It uses context as one of the driving forces in the planning process, and emphasizes on smart use of resources based on *supply and demand*. The ordering in which activities are executed, is based on resource availability. If the activities' interdependencies allows it, workflows may be processed in parallel, possibly distributed, to save execution time and exploit available resources.

In the previous chapter, we obtained necessary contextual information, made workflows, elaborated workflow adaption, and proposed how to manage knowledge of activities' context influence. In this chapter we will explain the components and planning process of the *Scheduling service*, where optimal ordering of workflows and use of resources is emphasised.

15.1 Context extensions

To be able to create a schedule, we need to know what will be the result of tasks executed and apply this in the process of planning ahead. As task execution incentives is context information, it is vital that contextual changes imposed by planned tasks are recorded to present the remaining tasks with a predicted context at a time in the future.

In order to do this, we propose *context extensions*. *Context extensions* are new, anticipated context that builds on the initial context. Hence, one context extension is a description which implies how the execution of a certain activity is expected to affect specific types of context in the real world, and thereby the foundation for simulating the outcome of the activity execution.

Figure 15.1 presents a timeline showing how the context is predicted to evolve over time. At t_0, or present time, we have a context c_0. This is the base context and *context resolution* at this point shows buffered context collected by a *Context service* (Section 6.1).

Fetching the base context by using the *Context service* could be a resource demanding

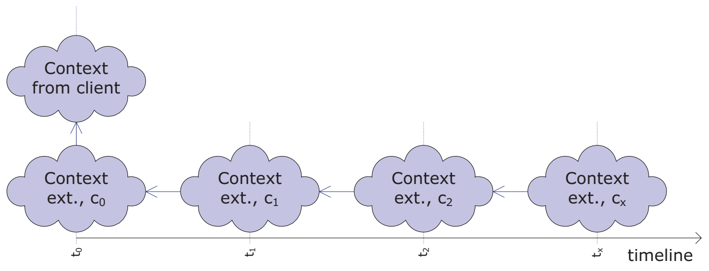


Figure 15.1: Context is resolved backwards through the context extensions and finally to the current context.

process depending of the amount of context to collect. More details can also increase the computational cost of decision making [17]. It is therefore important that only relevant context are gathered and that for every update, only requested context is gathered (this will be based on the constraints of the queued activities). In this way, c_0 only contains relevant context information with as little resource use as possible.

These *context extensions* are created by a *context builder*. It is placed in the architecture as shown in Figure 15.3 in the architectural overview of the scheduling service. The *context builder* component is responsible for creating a *context extensions* for each end of an activity in a schedule, as illustrated in Figure 15.5. It will build an *context extension* chain for each evaluated schedule.

Context information is fetched from the *context extensions* by a *context resolver*. If context information is not present, the *context resolver* will ask the previous *context extension* for this information. In this way, the *context resolver* backtracks trough context extensions, until the desired context is found or c_0 and the current context is reached without results. In this way, a chain of evolving context can be created and resolved using as little resources as possible.

15.2 Architectural overview

As a foundation for our approach in making the *Scheduling service*, we use the *Smart work processes architecture* (Section 6.1) to outline the service and get a better overview, hence delegate responsibility and plan how the service should connect with other services in the architecture.

Figure 15.2 shows the scheduling architecture located in the *Context-aware workflow enactment service* (CAWES) component of the *Smart work processes architecture*. CAWES is decomposed revealing the workflow, context, and enactment subsystems with their use of a task scheduler and supporting systems like *expert systems*, *CBR*, *reinforced learning*, and *empiric databases*. It also depicts the other important parts of the *Smart work processes architecture* in need of the scheduler.

Figure 15.3 shows the actual architecture of the scheduler. The *Context builder* is responsible for generating context as it is anticipated to be after execution of activities. A *Scoring resolver* will fetch and generate weighed scores for each task based on a dynamic set of criteria. (Examples of such criteria are stipulated in the *Scoring* chapter of [75].)

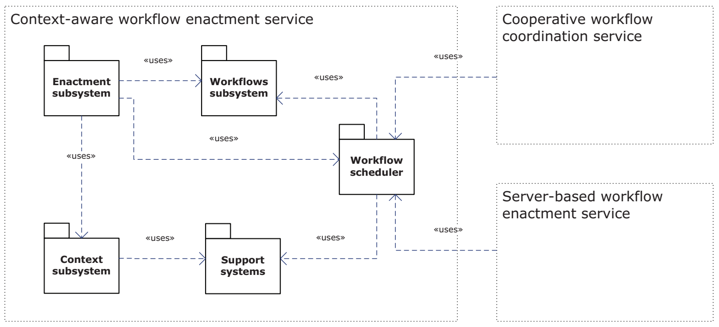

Context-aware workflow enactment service
Enactment subsystem
Workflows subsystem
Workflow scheduler
Context subsystem
Support systems
«uses»
Cooperative workflow coordination service
Server-based workflow enactment service

Figure 15.2: Scheduling placed in the context of *Smart work processes architecture*.

The scheduler then use the *Context builder* to generating alternative schedules based on contextual state, before it selects the most appropriate.

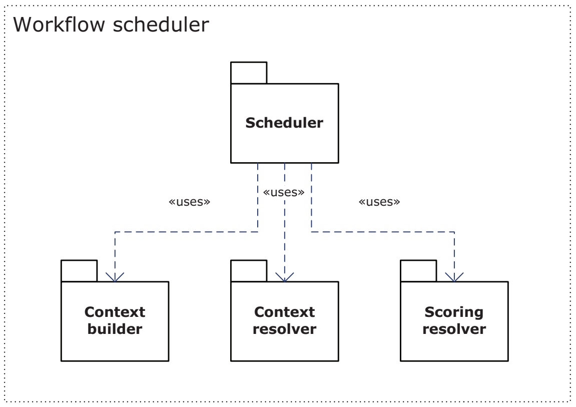


Figure 15.3: The architecture of the proposed scheduler.

While developing the architecture, we tried two different strategies of generating the schedule. The first strategy used an incrementally changing context where the change was made based on the simulated execution of a group of activities capable of being processed in parallel. An illustration of one of the plans and its context extensions can be found in Figure 15.4. This formed a planning tree made up of several branches because of all the different combinations of concurrent activities. One context extensions is created for each group of activities capable of concurrent execution in every branch, and this would have lead to major processing problems and unnecessary delayed execution times for some activities. The difference between the two approaches can be seen in Figure 15.4 and 15.5.

The second and better approach was less resource demanding and more dynamic, and therefore better supported *continually planning* (Section 5.5.3). Figure 15.5 shows a schedule put together by the second approach with context extensions.

The method for building the simulated context in the second approach, relies on the

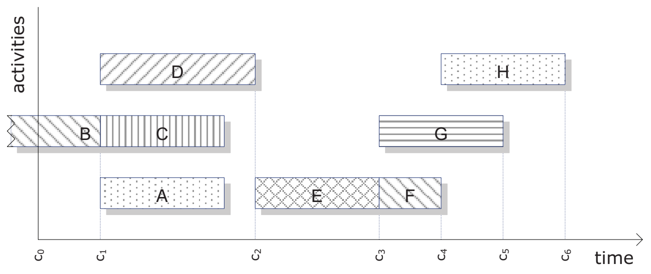


Figure 15.4: Expected context evolution over time for the first schedule approach.

estimated time of completion for each activity and not a group of concurrent activities. More *context extensions* are created in the second approach, but as seen in the Figures 15.4 and 15.5, it plans much more efficiant. Combined with the *scoring mechanism* to make a selection of activities to schedule next, the processing needs and utilitarian value of the schedule is greatly increased.

Before an activity may begin execution, the scheduled contextual foundation must be in place (the activity's constraints must be satisfied). Therefore, instead of viewing the schedule as levels of tasks, we will view it as levels of context which then can be matched against the activity constraints. An illustration of this can be seen in Figure 15.5.

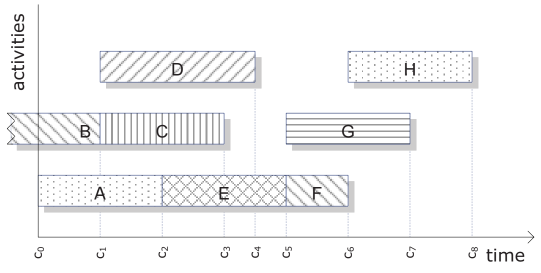


Figure 15.5: Expected context evolution over time for the second schedule approach.

15.3 The scheduling process

Scheduling will be performed with the steps presented in Figure 15.6 and elaborated in the following enumerated list.

Step 1. Build a context extension based on the current context.

Collecting all context information would be too much. This could be solved by collecting only the information requested by the tasks as necessary to execute or other information the task requests.

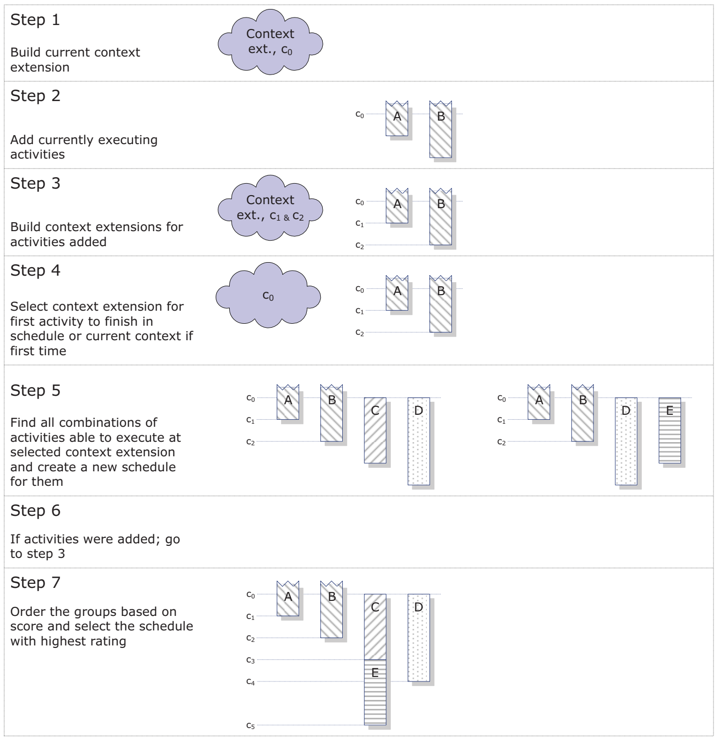

Step 1
Build current context extension
Context ext., c_0
Step 2
Add currently executing activities
c_0
A
B
Step 3
Build context extensions for activities added
Context ext., c_1 & c_2
c_0
c_1
c_2
Step 4
Select context extension for first activity to finish in schedule or current context if first time
c_0
Step 5
Find all combinations of activities able to execute at selected context extension and create a new schedule for them
C
D
E
Step 6
If activities were added; go to step 3
Step 7
Order the groups based on score and select the schedule with highest rating
c_3
c_4
c_5

Figure 15.6: Stepwise description of the scheduling process in the second schedule approach.

Step 2. Add the tasks to the schedule that already exists. All combinations of existing tasks, including no tasks, would be set up as schedules. This is to allow score calculation of the schedule with aim to stop execution of tasks if it is found most effective.

Step 3. Build context extensions for every task in each of the schedules. A context extension is an addition to the former context extension so that information is not duplicated in the system. How context for a given stage, e.g. c_1, is calculated is explained earlier in this Section.

Step 4. For each schedule, select the context extension for the next stage, that is the next task to finish. If you are at stage c_0 the next stage would be c_1.

Step 5. Find all combinations of tasks, which have satisfied execution criteria at the current context extension stage and that has no or manageable conflicts and create schedules for them.

Step 6. If tasks were added to schedules in the prior step, go to step 3.

Step 7. Calculate score for each schedule and select the one with highest score.

15.4 Markov decision processes

Planning under uncertainty is a central problem in the study of automated sequential decision making, and has been addressed by researchers in many different fields, including AI planning, decision analysis, operations research, control theory and economics. Many planning problems common in these fields can be modelled as *Markov decision processes* (MDPs) and analysed using the techniques of decision theory [17].

The scheduling process have some of the same characteristics as MDPs, so we make a comparison:

In MDPs, it is commonly assumed that a (context) state contains enough information to predict the next state. In other words, any information about the history of the system relevant to predicting its future is captured explicitly in the state itself. Formally, this assumption, the Markov assumption, says that knowledge of the present state renders information about the past irrelevant to making predictions about the future [17]. Hence, Markovian models reflects the fact that the present state is sufficient to predict future state evolution. This is the same assumption we make, when we gather information by using a *Context service* and then, in the *Context builder*, base our predictions of future context-states on this present state.

It is also common in MDPs to assume that the effects of an event depend only on the prevailing state, and not the stage or time at which the event occurs. This, however, is contradictory with our approach. We assume simulation of activity execution in a specified time as the foundation for predicting future context.

15.5 Problems and challenges

The proposition explained here, has some weak points. The scheduler may be very resource demanding if many activities exists in queue and these can form many possible compositions. The scheduling service could then profit from being hosted by a powerful peer, or when necessary being distributed through cyber foraging (Section 4.2.1).

If the context resolution for some reason is not able to resolve required context information, execution parameters may not be satisfied. Then predictions of context information must be manually entered, hence requiring user interaction with the client. This transformation of knowledge from the user to the client may be cumbersome. However, the greatest risk of this approach is that manually entered context are being kept and used by the scheduler, even if its outdated.

Scheduling workflows and activities based on resources on a location to which a moving entity is heading, can be tricky. This is however possible if the entity is connected to a widespread network or use a wireless technology with long reach. Estimated speed and route would be needed. The appropriate *Smart space* (location with ubiquitous services) could be found by deducting the location, using ontologies (Section 5.3.1). Then the needed resources can be asked for within this location, and scheduled as normal.

PART V

Discussion and conclusion

CHAPTER 16

Discussion

16.1 Context

The extended context definition provided by Kofod-Petersen and Mikalsen (based on Dey and Abowd [34]), is a good formal representation of context and interesting for our work. It describes context as domain independent, and match our use of context in the *Scheduling service* (Chapter 15) and *Arbitration service* (Part III).

Most formal definitions of context focus on the idea that context is some particular type of information. However, we serve *contextual information*, *information providers*, *actuating entities*, and *services* in the same component in our implementation, where they all are assembled into the term *resources*.

The open context model provided by Kofod-Petersen and Aamodt [55] is an interesting, domain independent categorisation of context from the user's perspective. It provides better understanding of how to implement and reason over context in a context-aware application by showing what types of context exist and how it can be arranged.

The infrastructure created by Henricksen et al. [50] is concerned with development of appropriate context modelling concepts for pervasive computing, and seems like a reasonable way to show the relations amongst context. It retains formality and generality and gives indications of information quality.

However, we have not been able to find any ready to use implementations of the above models, and therefore not used any of them in our prototype. Instead we have developed support only for the contextual information needed for our purpose.

The *Smart Work Processes* (SWP) architecture (Section 6.1) represents a generic framework for the ubiquitous paradigm with little limitations on what functionality to implement. It provides separation of concern regarding how to structure features in a context-aware application. It also retains coordination amongst entities (COWCS) and lays no restrictions on heterogeneity. These are all qualities we would like to preserve.

The *Java Context Awareness Framework* (JCAF) (Section 6.3) is a generic service infrastructure and programming framework for creating context-aware applications. The runtime infrastructure consists of three layers; a *context client layer*, a *context service layer*, and a *context sensor and actuator layer*. While the SWP architecture is a model for how to implement services in a ubiquitous environment, the JCAF actually provides a Java programming framework supporting some of the same ideas.

JCAF is a peer-to-peer context framework, which seems to support Dey's definition of context. We chose JCAF for our implementation because we thought it included a ready to use context framework, defining both the context model and lookup functions. However, it appeared that it did not satisfy our needs on some areas (Section 16.3).

The *Context Toolkit* (Section 6.2) is also an interesting architecture and programming framework. It is similar to the *SWP* architecture in many ways, but differs in some areas. One of the differences is the *Context widget* component's ability to store historical context. Widgets are public resources, which contain one context type only. They reside on servers in a distributed network. This could be a hurdle regarding mobile peer-to-peer networks which should have the ability of working disconnected from larger networks.

However, having a mechanism acting as an expert on the local context in a *Smart space* based on reasoning of historical context, could provide a considerable support when trying to predict how plans should be built, executed, and adapted. Hence, we will stick to this as a good idea.

One thing we find missing in the *Context toolkit* architecture is the ability to affect the environment by the use of actuators. This makes it a read-only tool for the use in context-aware applications.

Kofod-Petersen and Mikalsen [56] also argue that the *Context toolkit's* ability to reason about contextual information must be implemented for each domain and application. Thus, making it flexible only in the design and implementation phase, and not in run-time.

Baldauf et al. [9] indicates that the *Context toolkit* takes a step towards a peer-to-peer architecture, but that it needs a centralised discoverer where widgets, interpreters, and aggregators can be registered in order to be found by client applications. We agree in that a discoverer is needed. However, instead of a centralised approach, we rather propose a discoverer that works in a volatile, *Mobile Ad-hoc Network* (MANet) fashion.

Both the SWP architecture and JCAF use *Context services* (Section 14.1) to obtain relevant context from the environment. These are also similar to the *Context widget* described in the *Context Toolkit*.

The *Context service* provided in the SWP architecture are generalised, hence it does not provide a thoroughly described service, but a framework. However, according to the spesification, it provides all communication with sensors and it filters the information it receives founded on client-based context subscription (type and abstraction) to reduce the workload. We picture it to be available both public and personalised. The *Context services* residing in the public domain can provide context information from a *Smart space* to the user, while those locally on a client can provide personalised context to be used by services in the *Smart space*.

Rather than using one of the context models or context gathering services discussed, we have implemented a *Resource discoverer* in our implementation. The *Resource discoverer* is proactive in discovering resources in the environment. It provides and updates an overview of resources within e.g., a *Smart space* automatically. However, the service is not restricted to a defined spatial area, and resources can be found on request from a user within the whole network. Hence, we have not implemented restrictions on the number of allowed hops in the network, such as Bardram has done in JCAF's *Context service* and Satyanarayanan propose in [77]. The reason we have not done this is mainly the small scale implementation, which do not suffer from network congestion.

In our theoretical description of a *Context service* (Section 14.1), we have mainly mixed elements from SWP and JCAF.

Ontologies (Section 5.3.2) can be used to describe the meanings, or semantics of context, and thereby reason about contextual relations. The context infrastructure provided by Henricksen et al. do not seem to provide such posibilities. Here, entities are connected to attributes and other entities through associations, but we are not able to reason about, e.g., how related the attributes of different entities are (only a textual description is provided). We think it could be interesting to introduce ontologies into this model, to enable such reasoning techniques.

Creek seems to use an ontology network for describing all contextual information in a classification fashion. This also appears like an exciting idea, which reminds of the neurons and synapses in the brain.

SOCAM proposes an ontology-oriented approach to support context reasoning and context knowledge sharing. The middleware uses a central server to fetch context from distributed context providers. This context information is then not situated, which conflicts with the non-centralised properties of mobile peer-to-peer (Section 5.2.1). It also violates with Satyanarayanan's ideas of localised scalability in pervasive computing (Section 3.3).

CoBrA is an agent-based architecture for supporting context-aware systems in *Smart spaces*. Central to CoBrA is a server agent called context broker. Its role is to maintain a consistent model of context that can be shared by all computing entities in the environment. This is similar to what both the *Context widget* and *Context service* can provide.

A language that can describe ontologies is the *Ontology Web language* (OWL). A part of it, the *Resource description framework* (RDF), is used to write ontologies to files.

We describe resource types by classifications, through the use of ontologies written in the RDF language (Section 5.3.1 and 5.3.2).

16.2 Planning

According to Bardram [12] and Suchman [91], plans are a resource for guidance and in the making and execution of activities. They should be dynamic, continually adapting as activities are executed, reflecting successful work and contain suggested improvements. If these actions are fulfilled, a plan will be a highly reusable asset when scheduling workflows.

Activity theory emphasise on the connection between plans and the contextual conditions for realising these plans. Bardram discuss how plans are made out of situated actions, and in turn are realised in situ. He characterise this as *situated planning*.

Situated planning constitutes a complex research area, frequently addressed by the use of *multi-agent systems* (Section 5.2.3) to ensure cooperation in the process.

Agents (Section 5.2.3) could help a lot in the planning work. They could fetch other entities' plans and start planning on that basis. They could actually perform different parts of the plan, both to serve their host and to help others. If other actors change the context unfortunately, agents could change the context state back so that the initial plan still could be carried out (avoiding rescheduling).

Regarding the key concepts of *multiagent systems* and *service oriented computing* the following are taken into account:

- Ontologies are adopted to facilitate interoperation and dynamic reuse of artifacts such as activities and experience.
- Process modelling is the foundation for planning and scheduling, which is needed to allocate resources based on work and context states.
- The prototype workflow is adaptive because it can change dynamically due to contextual triggers.
- The service protocols we use (choreography) for entities (services) to interact, is based on JXTA (Section 6.4) and XML, which seems like a fair choice based on the open source and platform independent profile on the implementation.
- We have not used directories or facilitators because of the low scale implementation, where we are served just as well by letting entities interact in a *MANet* fashion.
- Quality-of-service measures are proposed to be a part of each activity, where different fidelities can be achieved through different stated constraints and success criteria.

However, we do not use agents when addressing some of the issues in this field. The *Scheduling service* (Chapter 15) make plans based on how future context will look like when combining already known workflows in different ways. From this, a plan is selected that combines context state and workflow execution in the best possible way.

This constitutes the foundation for setting up a best possible schedule for all activities with regards on total resource use in the relevant environment (*Smart space*). Resources are supposed to have their own schedules, but they do not necessarily have to be ”smart”, hence control which entities are granted access. This can be taken care of by an *Arbitration service* (Part III) where the involved parties agrees on a solution.

Bardram provides some guidelines for producing and altering plans. The workflow component in the prototype implementation (Part III) supports in situ construction and reconfiguration. This makes it ready for *adaptive workflow* techniques based on the dynamic environment, and thereby supports Bardram’s chain of thoughts about situated planning.

As a part in the process of obtaining workflows in an automatic fashion, we have come up with proposals for how to obtain activities based on only knowing what goal to achieve (Section 14.2).

By using the Markovian model’s assumption that the present state is sufficient to predict future state evolution, the *Scheduling service* represents an approximated situated planning approach through assumptions. As knowledge about how activities influence context grows better with learning, context predictions will also get better.

The scheduler is supposed to prevent conflicts and use resources in the best possible way. Nevertheless, when meeting a conflict state based on other actors interference in the environment, the *Arbitration service* is invoked. This service is implemented as our proof-of-concept prototype, and uses CBR in the mediate process to fetch and reuse previously solved conflicts. Ontologies are then used to beneficial exploit previously solutions which are similar to the new conflict, and therefore can be used directly or adapted to fit the new case.

We have not elaborated any specific solution in sharing of plans, but actors should know each other's plans to be able to adjust own plans accordingly and cooperate in executing activities and sharing results.

By continually revising and evolving plans and schedules according to the contextual conditions of the environment, our combined architecture supports *adaptive workflow* and *situated planning*.

16.3 Arbitration

We developed a proof-of-concept application to learn about different factors of arbitration between clients. We started out with our work from the *Architectural approaches supporting scheduling and arbitration of workflow processes in a cooperative environment* [75] prestudy. Here we outlined an architecture that supports scheduling and arbitration using artificial intelligence, but did not go into details of the inner workings and challenges of our approach.

Because of the shear size of the work implementing the arbitration service, we ended up limiting it to only this. The scheduler service has been updated with new theories, knowledge and ideas as these have come to us.

The challenge was then to produce a proof-of-concept implementation to learn more about the specifics of arbitration and to test if our architectural approach was viable.

First we created a peer-to-peer network of all devices running our implementation. Second, was to make context sources, sensors, and actuators available. None of the considered technologies for context-awareness were implemented. This was because the technologies did not have adequate support, were to complex for our use, or required too much integration code. We attempted to integrate JCAF without much success. The evaluation of JCAF can be found in Section 16.3. Since the *Resource package* (Section 12.5) could fill this role without much change, we decided to use this to achieve context-awareness instead.

Context information is classified with ontologies for type and location. This is the same approach as, amongst others, CoBrA and SOCAM have used. This greatly increases our applications ability to handle different implementations of a specific type, search, and reason around context sources. This is by us considered as a must for an arbitration service to be able to reason around resource requirements of workflows and conflicts.

Another finding was context integrity. When context information is created and maintained by fluctuating peers rather that stationary servers in one location, context frameworks might find themselves in a situation where two systems have defined the same entity with appurtenant information. When searching or reasoning on context, this appurtenant information must be merged, so that all context information is available in one deductive chain. Likewise, when one peer leaves the location, it must pass its discovered context information onto peers that still have interest in it. This is a challenge for context systems of today.

The biggest challenge we faced was to represent a conflict in a CBR case. After brainstorming on different representation, we found no obvious solution that were dynamic enough to represent all possible scenarios we came up with. The closest we came was the use of ontologies. For each case, both sides workflow, goal or subactivity that, when combined, results in a conflict, is represented by an ontology class or individual. This requires an ontology network depicting the nature of the activity to be created and passed with the workflow definitions.

Using this solution, we found that cases that are similar in nature could be detected even though the context snapshots were inherently different.

This presents us with both benefits and drawbacks. The most eminent drawbacks are that the ontology network must be well defined and that workflows must be tagged with the right individual. The benefits are that only two values are needed to store a representation of a conflict. It also has low storage costs, low search costs, and a high degree of accountability. We have effectively reduced the complex comparison of two dynamic tree structures with possibly different datatypes and structures, into comparing only the distance between two ontology individuals in a network.

Good as this is, it is not enough to say that a conflict is impending. It may be dependent on the state of context information. Lets use our room temperature scenario in Section 9.1. This conflict could be described by the ontology individual *Control_temperature*. If both actors in the scenario are in the same room there is a conflict, but if they are in neighbouring rooms there is no conflict. A conflict description must therefor take into account some other criterias to say that a conflict is present. This type of information is as complex as the description of a workflows nature, but not as easy to abstract into one ontology individual.

Our solution to this problem was to introduce *contextual* or *state triggers*. In our CBR cases there are two, namely *location* and *participants*. In the implementation description, the location trigger is called a filter. This is to describe the programmatic function and not introduce a new and possibly confusing term there. The list of triggers for each case should be dynamic and support nesting. Each trigger should have information on what context or state to look up, preferably by use of general domain knowledge, and how to compare it.

Our triggers were statically defined in the CBR case base as jColibri does not support dynamic case structures without comprehensive customisation. The *location trigger* was defined as a double ranging in the interval $[0, 1]$. This trigger concerns location nearness. Locations are, in our implementation, also defined by ontologies. The nearness of two places is defined as the distance in the ontology network. The value *1.0* means the same location, *0.75* means very close and so on. The trigger returns true if the location nearness of the conflicting parties is close, e.g. higher, than the one defined in the conflict description. This way you can ignore the conflict unless they are in the same location.

Triggers can be seen as conditions that either contribute to the similarity or excludes the case. Since many triggers may act as a set of rules we propose expert systems supporting fuzzy logic in combination with case based reasoning. The execution effective expert systems can filter out a set of cases based on their triggers that can later be searched by the CBR implementation. This would dramatically reduce the computational power needed to perform conflict retrieval. The fuzzy logic support renders the triggers of cases able to use fuzzy terms and reasoning during case retrieval.

When running the scenario in Section 9.1 (*Scenario 1*) we found that our implementation, as described in Chapter 12, was able to detect conflicts using the location trigger as intended. When the clients at a later time tried to control the temperature of a common but different location, it only informed of and performed arbitration on the conflict. This was because it already new about the conflict from running *Scenario 1*.

The use of ontologies to classify activities and transfer knowledge between domains were also tested. When we ran the scenario in Section 9.2 (*Scenario 2*) after *Scenario 1*, it found that the nature of the activities in the known conflict from *Scenario 1* were very close to the ones being currently executed in the room. It detected and informed the user of a possible conflict and a possible solution to it. When the solution was chosen, the triggers and solution of the known conflict was copied over to a new case with the activity classifications of the newly discovered conflict. This showed that is was able to predict and learn using general domain knowledge, as it was able to detect the conflict in *Scenario 2* only knowing about the conflict from *Scenario 1*. Using the implemented general arbitration strategy of adaption, it was also able to solve the newly discovered conflict and the newly learned case was then stored in persistency.

The scenario in Section 9.3 was not tested, but has been taken into account when designing the proof-of-concept implementation. In theory it should be able to solve the conflict by recognizing the conflict because it looks ahead in the workflow. Since the solutions are sensitive to the order of the actions involved in the conflict, it can always put one specific action on wait when in conflict with another. This means that it does not matter which forklift that discovers the conflict, the one outside the corridor will always wait as long as this is the desired solution. The second solution proposed in the scenario is not supported by our proof-of-concept implementation.

We have not developed a viable strategy for storing the solution to a conflict in our CBR case base. The reasons for this are several:

1. It was not the focus of our research.
2. We did not implement and test our solution with a fully developed workflow enactment system.
3. Good solutions will require advanced workflow adaption. A system that supports this along with the classifications of activities does not exist today and would be required to create a good solution structure.

However, we have seen that using ontologies to describe the solution, can be useful. It can then perform adaption on workflows without knowing anything about the workflows up front. Our adaption strategy merged workflows from two clients by taking the mean value of all numeric variables in the workflows with common identifier. This is a very basic and crude solution, but goes to show that some conflicts can be solved based on the mere nature of the solution strategy. We do not propose this as a final solution to the solution description, but combined with an XML adaption or solution description, it could resourcefully resolve complex problems.

The system has no problem choosing a solution when there is only one well suited. However, we believe that most of the time in real life, there will be many solutions to a problem. When several solutions are available for a conflict and the CBR system is unable to separate them, we need another way to select the best solution. For this we propose a scoring mechanism like the one presented in our depth study [75]. This scoring mechanism could take into account several factors like security, reasoning around the current situation, empirical use personal, cultural, and global, and other relevant factors. A total score of how relevant the solution is could then be the basis for a selection of a solution. This would relieve human interaction, which is highly desirable.

Requirements

Communication

F1 *Support an open communication standard, i.e., not be bound by hardware or proprietary standards.*
This requirement is fulfilled with the implementation of JXTA. This technology is open source and the API is abstracted away from hardware and protocols. It supports interoperability across different peer-to-peer systems and communities, is platform independent supporting multiple/diverse languages, systems, and networks, and is envisioned for every digital device.

F2 *Support ad-hoc, in-situ connecting and disconnecting of clients.*
This requirement is fulfilled by many components of the implementation. First, JXTA supports this nature, secondly all components that communicate with other peers are written so they cancel their current task that required communication with other clients, if the connection is lost.

F3 *Software components on a client should be addressable from the network.*
Each component that can receive messages, creates an input pipe. The pipe is named accordingly, so that other clients can search it up and address it directly.

F4 *The software should not be dependant on a communication service always being present.*
Some parts of the software are dependent in nature. But the none dependent parts, like the CBR subsystem, workflow enactment and logs work without a connection present.

Context

F5 *The system needs to be context aware.*
The system is made context-aware through the use of the *Resource package* described in Section 12.5. This is though not an ideal context awareness as this also has a lot of the weaknesses, described in the discussion on our proof-of-concept implementation, of JCAF.

F6 *Share context with other clients.*
This is supported through advertisement of the resource communication pipes. A resource is available to everyone who are members of the projects peer group.

F7 *Should use the communication component.*
Resources uses JXTA for communication.

Resources

F8 *Incorporate sensors, actuators and other knowledge information.*
Because of resources in practice is RMI over JXTA it is possible to implement nearly any component desired.

F9 *Should use the existing communication component.*
Resource uses JXTA for communication.

F10 *Distribution of resource information and/or discovery of resources in the environment.*
This is handled by JXTA. A resource creates a server pipe with a descriptive pipe name. The pipe advertisement is made searchable in the entire project peer group. All clients can connect and use the resource.

Workflow enactment

F11 *Enact basic workflows.*
Through the packages described in Section 12.6, a basic workflow structure and enactment is defined.

F12 *Handle start, pause, stop of workflows.*
These functions are implemented through the packages described in Section 12.6 on workflows, workflow enactment and scheduling.

F13 *Handle dynamic changes to workflows.*
A workflow can be dynamically changed during execution without stopping or pausing it. Functionality for each *ExecutableActivity* and *Statement* can be added, removed or changed.

F14 *Monitoring of workflows executed remotely.*
This is implemented through the *RemoteWorkflowServer* and *RemoteWorkflowConnector* classes.

F15 *Read and manipulate resources.*
This functionality is implemented in the statements *FindResource* and *UseResource* described in Section 12.6.

Arbitration

F16 *Find similar conflicts in the repository.*

This is achieved through jColibri in combination with ontology classifications with customized similarity methods and the filters, also called triggers, described in Section 12.7.

F17 *Persistent repository.*
This requirement is fulfilled by jColibri with our customisation of the *Connector*.

F18 *Order similar conflicts according to the degree of similarity.*
Done by the similarity function of jColibri and our customizations. A value in the interval $[1, 0]$ describes the degree of similarity.

F19 *Adapt prior solutions to new conflicts and store them in the repository.*
This is handled by the *Arbitration* class. It will create a new instance of a conflict-solution case and copy the new conflict description to that. This case is then stored, and thereby learned.

F20 *Call for solution if none is found.*
This requirement is not implemented in our proof-of-concept application. New conflicts, not a result of an adaption, must be manually entered in the conflict repository.

F21 *Be able to discover conflicts between other clients and itself.*
This requirement is achieved in combination of the *ConflictDiscoverer* and *ArbitrationCBR* class, as described in Section 12.7.

F22 *Negotiate which client is responsible for the arbitration.*
This has not been implemented. In our implementation, the client that discoverers the conflict will control the arbitration process.

F23 *Make use of other clients knowledge.*
This is achieved by sharing solutions to conflicts that arises. Either if it is already defined or if it is a newly discovered and adapted conflict. They can also use the knowledge from resources residing on another client.

F24 *Distribute solution to participants so that they also learn.*
This is done during the arbitration process.

F25 *Must be able to remove, add or adapt workflows in the workflow enactment component.*
The *Arbitration* class has full access to the workflow enactment subsystem, and is able to add, remove and adapt workflows.

Technology evaluation

JXTA

We started the implementation by setting up JXTA and creating a peer discovery service that kept track of all peers that were connected to our peer group. Setting up JXTA is quite complex but good tutorials exist. Since our peer group was open (without any security) it was fairly easy to connect.

During implementation of the peer discovery service, a problem regarding caching of advertisements occurred. JXTA caches all found advertisements and passes them on to the other peer when a discovery request is received.

This results in that advertisements from peers that are no longer part of the network is still returned through other peers on the network.

We wanted to keep an updated list of all peers currently on the network. The list should have the property of keeping adding a peer when found, keeping it in the list while connected, and remove it when it is disconnected.

Since the peers do not inform others when they disconnect we needed to create a list that was synchronized with periodically request for peer advertisements.

To overcome the problem of caching of peers, we needed to flush all peer advertisements from the local repository. The implemented method of flushing all advertisements of one type did not seem to work as expected.

The solution was to remove all advertisements of remote peers, locally, after they were discovered and synchronized with the list. This way, only the advertisement of our local peer is sent to others when a request is received.

Each entry in the peer list, has a timeout variable that hold the count of discovery requests that have been sent since the last advertisement was received. If this passed a defined amount, it will assume that the peer has disconnected and remove it from the list.

The same problem occurred for pipe advertisements but was a known situation at that time and therefor easy to overcome.

To set up a pipe was an easy affair that only needed a few lines of code. After fiddling around with peer advertisements, and discovering a solution, it was also easy to look up pipe advertisements and keep these advertisements up to date. Four types of pipes were implemented. Input pipe, output pipe, server pipe and bidirectional pipe. All were easily implemented, but especially easy bidirectional communication was achieved with the combination of JxtaServerPipe and JxtaBiDiPipe.

One of the great features of JXTA is that it is implemented for both J2SE, J2ME and C/C++/C#. This allows a greater diversity of clients and other implementations. Combined with the ability to form peer-to-peer networks over Internet and through firewalls, along with protocol and carrier independencies, makes it a powerful tool.

JCAF

This looked as the optimal solution. A context aware framework for Java, developed by one of the leading scientist in the field of context, Jakob E. Bardram. After some time we found the framework not suited for our implementation. This is based on several different factors.

When implementing the solution, we would have to make some work-arounds to make the framework suitable. Although a fairly good basic tutorial was available, the Javadoc was almost not existing and very poorly written. This made the workarounds more cumbersome.

To implement and use the program from within our code was very easy. With the supplied tutorial it took only a few lines of code to get started.

JCAF supports linking several context sources together. It is built upon the thought that there are some context servers and many consumers. These servers can be interconnected explicitly via JavaRMI using IP addresses. This practise is in contrast to our implementation that needs context to be maintained in a peer-to-peer fashion, without dedicated servers.

JXTA is designed to communicate over several different communication technologies and protocols. To implement context sources that depend on the IP protocol was therefor not desirable.

JCAF does not support pluggable communication components. This would have solved the two drawbacks mentioned above.

The framework supports adding listeners to entities in context. This could either be done by adding a listener to a single entity or adding a listener to the context server for all entities. It will fire events for changes in the entities.

What this listener service greatly lacks, is the possibility to add a listener to the actual context server, possibly with filters, to be informed of additions, changes, and removals of entities. The add listener method thought to do this only adds to the entities currently present in the context. No events are fired when a new entity is added.

If your application uses context information, it needs to be able to search all context sources for the desired information or in some way deduce it from other entities.

There exists only one method for looking up context information in JCAF. This uses only the id of the entity as a parameter and does not, as far as we know, support wildcards. Discovery of prior unknown sources and types of context is therefor very cumbersome. This severely reduces the utilitarian value of the whole context framework, and is by us considered to render JCAF unusable in our implementation. A workaround could be implemented, but the solution would be much less than ideal as the search would be implemented over JXTA, resulting in context being managed, searched and used over both RMI and JXTA.

This framework has a very good representation of context but due to our need for searchable context sources rather that relations between context elements, the decision to implement context as *Resources* (See Section 12.5) was made.

jColibri

jColibri comes with Javadoc documentation. This is incomplete as only some methods are documented.

On their web cite there are some video tutorials that do help you get started, even though these, as the Javadoc, seems incomplete. The best method to learn how to use jColibri ended up to be the *try and fail*-method.

We could not find any instructions on how to implement jColibri in our application. From the jColibri graphical interface you could set up the cycles and export it to Java code. This java code did not let itself start with our implementation without errors. There were a lot of configuration files and classes that needed to be copied to our project folder. We ended up copying all code and configuration folders from jColibri. This meant that the extra components and examples also followed. Some editing of the exported java file was also needed to get the paths correct.

jColibri should come with description on how to implement it as it clearly is intended also to be used in stand-alone java applications.

When jColibri finally was implemented and started there was time to tweak it to fit our application. This showed to be more complicated that expected. The configuration of jColibri is an endless list of adding statements to the configuration that you will run later on. These statements were not added as java object instances but as complete string class references for jColibri to instantiate. All this code was placed in one method and was hard to get on top of.

The modifiability of jColibri is very good. Every task and statements of every step is modifiable.

jColibri supports the use of ontologies through Jena. The standard connector should be able to write and read ontology individuals to and from persistent storage. Nevertheless, it did not, reason unknown, manage to restore the individuals from storage. No error messages were seen, and a solution not found. We therefor ended up writing our own connector.

To achieve the desired functionality, we needed to exchange some of the tasks and statements with our own. Many extensions were written before we ended up with using only the following four.

We had to write our own connector. It is the component that reads and writes the case base to persistent storage. The reason was problems with reading of ontology individuals.

The out of the box methods for calculating the similarity of two ontologies did not function as expected. This was possibly because a case has two individuals from the same tree. This is a restriction in jColibri, possibly not intentionally. It required us to copy the classes and individuals in the ontology tree to two separate root nodes. Fortunately, this can be done in RDF by only referencing so that changes are reflected in both branches.

After this the similarity function still did not functions as expected. We therefor ended up writing a task to calculate the similarity between to ontology individuals. The description of this task can be found in Section 12.7.2.

The properties that jColibri calculates the similarity from is constructed as a tree. The default method of jColibri is to calculate the mean value for each element in a node, and propagate up to one single mean value of the top of the tree. This methods excludes the possibility of having a value that defines the case as not similar, independent of the other values. We could have created our own task for this merging of leaf values as well, but ended up inserting a filter in the chain of tasks to achieve our goal. The similarity tasks of location and participants was therefor implemented always returning 1.0.

jColibri was clearly implemented to perform a complete cycle on each call. The jColibri instance has only one context. In this context all the operations on cases are done. It first loads all cases into context, calculates similarity, chooses the best matches, performs adaption using the query object also in context, before it performs retention. During all these tasks the *working cases*-list is updated. Since we have a multi-thread model, there is a great chance that one thread is in the process of adapting the cases in the *working cases*-list whilst another thread clears the context and loads all the old cases. The access to jColibri therefor had to become thread-safe.

This was done, in a somewhat clumsy, but the only viable way. Since our implementation could not guarantee that a cycle is completed without breaks for user input and slow communication lines, the cycle had to be broken up into pieces, letting each thread execute only parts of the cycle. This was achieved by bypassing jColibri on some things, like adaption of cases. Each synchronized access point to a part of the cycle clears the context and prepares it, by restoring working cases and query before it continues the execution of that cycles.

This is considered by us as a great weakness of jColibri. A separate context should be created for each invocation of a cycle, so that it easily can be used in a multi-threaded application.

jColibri has it own definition of a case and query. These objects have a structure where you can set key and value pairs to create a tree of properties. To read values out of these objects required several lines of code, was cumbersome and provided no type security.

Most programs use an internal object to represent a case and solutions. We had to write our own converter between jColibri's case representation and our own.

Since so much of the CBR-cycle needs to be customized, jColibri could consider to support user created objects to be better integrated with the application using it.

Jena

Jena comes as a part of and is instantiated and configured with jColibri. We have not discovered any disadvantages with Jena and have only positive feedback about the ease of use and good overview of this component.

16.4 Research questions

RQ1

How can we represent a conflict between two clients in a dynamic and generic way in a Case-based Reasoning case repository, so it can:

RQ1.1

Recognize the same conflict in a different context?

Storing a representation of a situation is not too difficult. The problem lies within filtering out the important information and storing it. This should not only be done to save storage space, but mostly because some contextual information is not important and therefore can hinder the situation from being recognized. E.g., you want to recognise a *control temperature* conflict regardless of a chair's position in the room.

The selection of these properties, or contextual elements is subject to constant research and is not a part of our study. We only see how this information can be stored in a case repository when it has been identified.

Our solution is to represent the nature of the conflicting workflows or activities as ontologies in the case-base. By doing this we are able to predict conflicts in workflows or activities of similar nature.

In the case-base, the contextual states that identify a conflict are stored as a tree. These are called contextual or state triggers, as they may check for more than just context. Triggers must contain the general domain knowledge of the context entity so they can reason for interchangeable context entities and the state that must be satisfied.

An *expert system* should sit between the case repository and the main CBR cycle. It should check contextual and state triggers and filter the list of cases to achieve efficiency. The computer-intensive process of CBR similarity-computation could be limited to a minimum by being used in combination with the very cost effective expert system.

RQ1.2

Predict a possible conflict that is similar in nature to ones that are known?

The proposed answer to this question also include the use of ontologies. The workflows' or activities' classifications that make up the conflict description depict their nature. Classifications in proximity should encounter the same difficulties in concurrent and cooperative situations.

As a result of the classifications, it is possible to derive and possible foresee problems that are not composed of the same workflows or activities as the ones known. Using this technique, the system is able to learn a new unknown conflict by itself, adapt, and invoke a solution.

There is a great challenge to this approach. That is to keep the general domain knowledge consistent, correct and maintain the semantic quality in a network where changes can be made by anyone.

RQ2

How can we support in situ planning of work processes with minimal user interference?

In the quest of a system which requires a minimum of interference from the user when planning and executing work, a technique must be elaborated that can automate this based on available knowledge and information.

Our prerequisites in the elaboration of this question are that workflows are made up by a chain of activities, the motivation for a workflow is to achieve a goal, and that workflows are the constituent parts of a plan. We derive the following subquestions:

RQ2.1

What mechanisms are needed to automatically construct workflows with only the desired goal as a starting point?

Basic criteria for being able to address this question is the presence of contextual information to base decisions upon, and a method to obtain such information. The requirement of minimal user interference indicates that activities must be obtained from some sort of knowledge or experience base. An activity must then have certain attributes that makes it possible to filter relevant activities from those not suited for achieving the goal.

We believe that one attribute should contain information of contextual requirements that must be satisfied before the activity can be executed. Another attribute should be success criteria, which give us an understanding of what the activity aims to achieve. Provided fidelity is also be relevant where quality is important for fulfilling the goal. As activities are reusable assets, it is also of interest to know about how well an activity accomplish its function.

With access to this information, and available activities, we have the basics for finding and adapting the activities that best suits the contextual situation and results in achievement of the workflow's goal.

RQ2.2

How can existing technologies be combined to support automatic workflow construction?

Gathering contextual information is proposed done by a *Context service* (Section 14.1), which is based on already existing research.

To be able to provide the mentioned attributes in an activity description, we have ended up with a structure based on the *Resource description framework* (Section 5.3.1 and 5.3.2). This language supports *ontologies* (Section 5.3.2), which allows measuring of hierarchical relationships with other activities.

How an activity has actually performed and what results it has delivered, can be measured by using *Reinforcement learning* (Section 5.4.5). This information should probably be included into the activity description along with the other attributes. Such information could then be used in finding the best suitable activity amongst similar activities.

The actual proposition for finding suitable activities when assembling a workflow is based on *Case-based reasoning*. By saving experience about which activities best satisfies what goal, makes us capable of reusing whole workflows. If no complete workflows exists that satisfy the goal, it is possible to construct a workflow, based on activitie's attribute values, by choosing activities that constitutes the difference in current situation and the goal's desired situation.

RQ2.3

How can workflows be made adaptive based on context?

Situated planning as described by Bardram [12], is a consequence of actions performed *in situ* which changes the contextual conditions.

To keep a workflow situated (executable in the current situation) while context changes, it needs to be adaptive. An adaptive workflow must be able to exchange one or more of the activities that represents its constituent parts, so that constraints are satisfied.

By using the mechanisms and technology refered to in *RQ2.1* and *RQ2.2*, it should be possible to obtain better suited activities and insert these to the relevant workflow. However, it is not desirable with workflow adaption whenever a contextual parameter changes. To make workflows more robust, we suggest the use of *Expert systems* (Section 5.4.2) with *Fuzzy logic* (Section 5.4.4) when interpreting an activity's constraints and success criteria.

However, exceptions in the process will occur from time to time because of contextual change. For obtaining substitute activities we suggest *case-based reasoning* or *soft case-based reasoning* which have better capabilities and include *fuzzy logic* amongst other techniques.

RQ2.4

How to ensure minimal replanning?

Replanning takes place when the current plan cannot execute due to context conditions. Reordering and finding combinations of workflows that works, or re-constructing the relevant workflows are then alternatives for getting a functional plan.

However, such replanning is resource demanding, and should therefore be kept at a minimum. We suggest an alternative approach, the *Scheduling service*, where contextual state are simulated based on activities' assumed influence on context (Chapter 15). Information of an activity's context influence should also be in its description, as stipulated in *RQ2.1*. The creation and improvement of such information could be taken care of by *Artificial neural nets* (Section 5.4.6) and *Reinforcement learning* (Section 5.4.5).

Another means to achieve less replanning is to avoid conflicts. If a conflict occur amongst actors, replanning is most likely an outcome for one or more parties. One way of preventing conflicts, is to share schedules with other entities within the perimeter of the relevant environment.. By doing this, other planning services can take these schedules into account when planning. With many actors within an area, this could however lead to massive distribution of schedules, and diminish the value of the approach. Letting resources keep an up to date schedule for when they are available and when they are booked by some entity, seems like a more reasonable approach.

When conflicts arise, they need to be solved. The negotiation process is conducted by the proposed and implemented *Arbitration service* (Part III). How this solves conflicts will also affect the need of replanning.

16.5 Research methods

Here we will evaluate our research methods. Figure 2.1 shows how the different methods have been used during our project.

RM1 *Literature survey*

We have used *RM1* to read up on theories and technologies that will help us understand the problem domain. Doing this, we have gained valuable knowledge that has helped us construct our research questions and thereby also helped us form our scenarios. During our project, this method has been used several times to gain knowledge on newly discovered topics.

RM2 *Descriptive study*

This method has been used to go deeper into the problem domain and develop it further. When selecting our scenarios, we used *RM2* togheter with the results of *RM1* to identify what key points the scenario had to contain.

Using a descriptive study, we grew more familiar with the state-of-the-art from our researchfield. This helped us form our problem definiton, which again was a guidance for our research questions.

This method also directly contributed to the making of our own contribution and conclusion.

RM3 *Analytical study*

Having obtained a deeper knowledge of the domain in question and formed research questions, *RM3* was used to break our findings into more understandable pieces. We then restructured these and added our own thoughts to the work.

During the implementation phase, *analytical study* contributed to the decisions that were taken.

RM4 *Scenario building*

We have created scenarios to help the readers, and our selves, understand how the system would ideally work and the problems that have to be overcome.

The scenarios were the foundation for the planning, architecural design, requirements and development of our proof-of-concept implementation.

RM5 *Requirements elicitation – Greenfield engineering*

RM5 was used to extract requirements from the constructed scenarios and the prior work from our prestudy [75].

CHAPTER 17

Conclusion

In this chapter we summarise the most important techniques and results provided in the report. We chose to present these by the two main parts, which is *planning* and *arbitration*. Because these are somehow interconnected, a slight overlapping takes place while describing some topics.

Planning

Planning of activities with respect to the current context, or *situated planning* as described by Bardram [12], is a dynamic and intricate process. We have divided this area into several parts to make it more manageable.

The basic challenge is to obtain the current environmental context. This is the foundation for reasoning about how planning of work processes should change with the environmental state. For this purpose, we have elaborated a *Context service* (Section 14.1) derived from previous research conducted mainly by Sørensen et al. (SWP) [88, 89], Dey (Context Toolkit) [33, 35], and Bardram (JCAF) [14].

A comparison of components from these frameworks has made us able to merge what we consider the best solutions in the different approaches. This makes us able to provide one solution that better suits our comprehension of such a service's responsibilities. We believe important roles for a *Context service* is to provide only relevant context information, with the correct abstraction, to the right user. In addition, we believe that a *Context service* should be able to reside both locally on a client (serving personalised context) and in a *Smart space* (serving environmental context). Storing of historical context is also considered for reasoning purposes, however, it should not be stored uncritical due to resource constraints.

The resulting *Context service* architecture is more concrete than the one described in the SWP architecture, but more conceptual than the one in implemented in JCAF. The idea of storing historical context for reasoning purposes is fetched from the Context toolkit, but not represented in the architecture. Because an implementation of the *Context service* has not been carried out, we lack information of how suitable the proposed architecture is for the intended area of application. Techniques for achieving the suggested functionality does however exist.

Activities make up the constituent parts in a work process, and can be put together to achieve a goal (achieve a certain change in context). We propose that an activity should contain clearly defined *actions* (what it will do to achieve its part of the goal), *constraints* (preconditions for when certain tasks can or cannot happen), *success criteria* (postconditions for when a task is fulfilled), *fidelity* (the activity's QoS), *contextual influence* (how an activity are anticipated to affect context), *scoring values* (to help in arbitration decisions), and *ontologies* describing different parts of arbitration proceedings (classifications of the activity's nature, regarding e.g. conflicts and problem solving). Activities can be described by using XML.

The information residing in an activity description is vital for its user. The properties mentioned here are those we find interesting regarding our work and are used by both the planning and reasoning mechanisms we describe. Many other properties could possibly be included to support other functionality in other services. An activity's amount of inherent information directly affects network congestion when exchanged amongst peers. If activity properties are to be stored in the actual activity description or other places like on centralised servers, is not elaborated. What is important, is that such information is always obtainable and linked to its respective activity so that activities can be reusable assets no matter where they reside in the environment.

The motivation for a workflow is represented by a goal. A workflow is further built up by activities that are able to achieve this goal when executed. In order to automate the creation of workflows, the workflows' constituent activities needs to be obtained. From this, a requirement follows that activities are suited for reaching the goal in the relevant contextual situation.

The approach we have chosen for finding suitable workflows based on a goal, includes several techniques. One of these is *case-based reasoning*. This learning and reasoning mechanism is able to find the most appropriate solutions from a case-base of previously created workflows. However, this approach requires that a suitable workflow has been previously provided.

If suitable workflows are not available through this approach, alternative solutions can be provided by building workflows from single activities (derived from other workflows). By using information provided in activity descriptions (described above), it is possible to find a combination of available activities which satisfies a workflow's goal and the contextual situation.

Ontologies provide the possibility of relating activities to each other based on classification, making it easier to compare and find the activities that are needed. *Reinforcement learning* could be used to rate workflows and activities based on their performance, which then could be yet another information property in an activity description.

The success of an approach to automatically build workflows is rooted in how to represent an activity to make it publicly applicable for this purpose. We believe this is an important challenge to be addressed.

Actions taken in an environment affects the context. Changes in context may prevent already generated workflows from executing. *Workflow adaption* is then the process where activities in a workflow are modified or exchanged to suit the new contextual state.

In order to minimise the need for adaption, we suggest that *expert systems* along with *fuzzy logic* is used when deciding if the contextual state is out of bounds compared to the activity constraints. This can make the system more tolerant to changes, hence less prone to exceptions.

However, when a workflow is no longer capable of execution due to changes in context, it should adapt to the new situation. For this we suggest to use the same approach as when automatically fetching activities to build workflows. The weakness in this approach becomes visible when no good substitutes exists. Then suitable activities must be added manually by the user.

When workflows are generated and up to date, they are assembled into a *plan* in a way that best utilises resources by the *Scheduling service* (Chapter 15). Information about how context will change in the environment can be obtained based on the knowledge of how other actors plan on using resources. This knowledge can be obtained from the relevant resources by requesting their schedules. When knowing which activities that will be executed in the environment in a certain time span, each activity's assumed context influence can be used to build a total *context extension*. This makes it possible to plan ahead in a *situated planning* manner.

One weakness with how this whole planning process is suggested done, is the demand for computational power when many workflows are to be planned and activities obtained. In addition, continually updating other actor's schedules to avoid conflicts and ensure correct context simulation, can be network demanding and then very power consuming.

Arbitration

We set out to find a way to represent a conflict between two clients in a dynamic and generic way in a case-based reasoning case repository. The complexity of a situation makes this difficult as a system is unable to know which information is important to recognise a conflict. If information is stored about the conflict that is not comparable between the different instances of the same conflict, it is impossible for the system to recognise it. Another problem is the computational power required to perform continually comparison of all cases, in constantly growing case repository, to all situations a client is involved with.

We found that using *ontologies* to describe the nature of an activity or goal, together with *contextual* and *state triggers*, renders the system able to recognise a known conflict and predict a similar unknown conflict. By running the scenario in Section 9.2 after the scenario in Section 9.1, our implementation discovered a conflict not prior known. This was a direct result of the ontologies depicting the nature of the activities.

Although not implemented in our proof-of-concept implementation, we propose using an *expert system* with *fuzzy reasoning* capabilities to perform initial filtering of cases based on the defined *triggers* to dramatically decrease the systems computational requirements.

Each case is described by two ontologies. These depict the nature of the goals or activities that were the cause of the conflict. The requirements for the ontology network is that each individual in the network describes an action or goal, and that the nature of the actions and goals is deductible from it.

When workflows are defined, the correct individual must be associated with the activities and goals that constitute the workflow. By having this information, a system is able to reason around the nature of the workflow it is executing and easier detect and predict conflicts and choose solutions.

To match a perceived situation against the cases in the repository is then comprised of calculating the distance of nature between the matching individuals describing the current perceived situation and the known conflict. The result is a percentage depicting the resemblance.

This alone is not enough to constitute a conflict. It may be that the two activities or goals must be executed in the vicinity of each other to be in a state of conflict. This is achieved by the *contextual* and *state triggers*. A *trigger* is intended to either inform that the described conflict can not occur because of the context situation, or contribute to the calculation of the case similarity.

We propose that *triggers* can perform a variety of operations and are organised as a dynamic tree construct. Their nature could be like the static location trigger implemented in our proof-of-concept application. It filters out cases that does not fulfil the demand of workflow execution proximity.

Because of the rule nature of *triggers*, we furthermore suggest that an *expert system* filters the list of cases based on *triggers* before the CBR system performs similarity calculations. This way, the computational effective expert systems can filter cases that, because of the *triggers*, can not occur. This leaves the CBR system with only performing similarity calculations on a reduced set of cases, dramatically improving the conflict searching process.

When the system learns more and more conflicts and solutions to them, there most likely will be more than one solution for each conflict. The question is then to select the best solution for that particular situation. We propose using a scoring mechanism to solve this problem. The scoring mechanism should take into account several factors like security, reasoning around the current situation, empirical use of solution both personal, cultural, and global, and other relevant factors. A total score on the relevance of the solution is then decisive for the selection of a solution. This would relieve human interaction, which is highly desirable.

CHAPTER 18

Further work

Pervasive computing is a very broad research area where much work is yet to be done. A lot of work done in this field has focused on providing different architectures and frameworks on how to best interact with the environment in a pervasive way. These approaches have been based on ontologies, context reasoning, distributed proxy servers, scarce resources in mobile device and so on.

Our contribution has been split into an *arbitration* part, focused on finding a way to store conflict representations in a CBR repository, and a *planning* part, elaborating techniques for achieving automatic situated planning and adaption of work processes. We will now present future work that is needed to improve on our findings.

General

- Keep the general domain knowledge consistent, correct and maintain the semantic quality in a network where changes can be made by any.

- Create a context framework that supports lookup of entities maintained on several peers in a mobile ad-hoc network.

Arbitration

There is much more to be done in the area of arbitration. Our work focuses on the discovery of conflicts and not the solution. Although it seems like a good idea, much more research must be done on *context* and *state triggers* to say if it is a viable solution.

Construct solutions
How do people communicate solutions to a device? Is the best way in the form of a discussion? Could the device store the discussion between conflicting parties and later use that discussion as a basis for resolving a similar conflict?

Select solutions
Many known conflicts may match a situation and have different solutions. How can we choose the best amongst these.

Case adaptation
When the device has learned a way to solve a specific problem, how can it adapt the solution to other problems? Will this be possible without human intervention?

Triggers
Further elaborate the viability of *context* and *state triggers.*

Expert systems Investigate if the potential gain of filtering the case-base with a expert system before passed on to a CBR system.

Large scale tests
Perform large scaled tests of the proof-of-concept implementation. How does it react when there are many cases in the repository, many peers, and many and large workflows?

Planning

We think one of the success criteria for a general context-aware system is how easy new workflows can be defined and executed in order to achieve the user's goal. Another criteria is the quality provided to the user when addressing the goal. The following points are suggestions of what should be pursued and improved within the planning domain presented in this thesis.

Implement the provided planning approach
We have in this report given a more or less specified set of ideas regarding the whole lifecycle of obtaining context and work descriptions, for then to plan how to perform the work and schedule resources for execution. A step towards a generic, context-aware system would be to create an implementation of the mechanisms described, possibly using and extending the service-oriented context-aware platform described in this thesis.

Prediction of context states
We describe a method for how to simulate future context for better planning purposes. This method is based on context influence information connected to each activity, and assembled into context extensions. An implementation which investigates this technique further are needed in order to improve it and find out if it is a feasible approach.

Evolve the workflow scope
The CBR based approach for finding suitable workflows requires that similar workflows actually exists in the case base. How can we automatically generate new useful workflows based on already existing workflows in a sparse workflow-base, relieving users from the input job and drastically improve deployment time and system utility? Filtering mechanisms for preserving a good QoS must also be developed. This implies to keep the case-base at a reasonable level.

Provide better and faster results
The proposed approach for constructing workflows based on single activities requires

that a suitable activity description is provided. What this description should contain on a general basis, and how this information should be obtained, still needs to bee addressed in order to create good solutions. In addition, solutions are needed that promote the use of effective expert systems as filtering mechanism, while focusing on quality when comparing and selecting similar activities.

PART VI

Appendix

APPENDIX A

Resource DTD

The DTD, Document Type Definition, describing a resource.

```
<!ELEMENT resource (type,description,location,method+)>
<!ELEMENT type (#PCDATA)>
<!ELEMENT description (#PCDATA)>
<!ELEMENT location (#PCDATA)>
<!ELEMENT method (parameter*)>
<!ELEMENT parameter EMPTY>

<!ATTLIST method name CDATA #REQUIRED>
<!ATTLIST method returnType CDATA #REQUIRED>
<!ATTLIST parameter type CDATA #REQUIRED>
```

An example XML file of a resource description.

```
<!DOCTYPE resource SYSTEM "resource.dtd">

<resource>

  <type>Temperature_sensor</type>
  <description>A temperature sensor.</description>
  <location>White_room</location>

  <method name="getTemperature" returnType="double"/>
  <method name="setTemperature" returnType="void">
    <parameter type="double"/>
  </method>

</resource>
```

APPENDIX B

Java resource description example

An example how to write the code that creates an resource description xml like the example of Appendix A.

```java
ResourceDescription resourceDescription;
resourceDescription = new ResourceDescription();

// Sets the popular description of the resource
resourceDescription.setDescription("A temperature sensor.");

// Set the ontology type of the resource
resourceDescription.setType(
        Framework.ontologies.
        getInstanceByLocalName("Temperature_sensor")
);
// Set the ontology based location of the resource
resourceDescription.setLocation(
        Framework.ontologies.getInstanceByLocalName("White_room")
);

// Add a method for getting the current temperature
ResourceMethod getTemperatureMethod = new ResourceMethod();
getTemperatureMethod.setMethodName("getTemperature");
getTemperatureMethod.setReturnType("double");
resourceDescription.addMethod(getTemperatureMethod);
// Add a method for setting the temperature
ResourceMethod setTemperatureMethod = new ResourceMethod();
setTemperatureMethod.setMethodName("setTemperature");
setTemperatureMethod.setReturnType("void");
setTemperatureMethod.addParameter("double");
resourceDescription.addMethod(setTemperatureMethod);
```

APPENDIX C

Invoke methods DTD

The DTD, Document Type Definition, describing the request to invoke methods.

```
<!ELEMENT invokeMethods (method+)>
<!ELEMENT method (parameter*)>
<!ELEMENT parameter EMPTY>

<!ATTLIST method name CDATA #REQUIRED>
<!ATTLIST parameter value CDATA #REQUIRED>
```

An example XML file of a request to invoke methods.

```
<!DOCTYPE resource SYSTEM "invokeMethods.dtd">

<invokeMethods>
  <method name="setTemperature">
    <parameter value="21.3"/>
  </method>
  <method name="getTemperature"/>
</invokeMethods>
```

APPENDIX D

Invoked methods result DTD

The DTD, Document Type Definition, describing the results of method invocations.

```
<!ELEMENT invokedMethodsResult (method+)>
<!ELEMENT method EMPTY>

<!ATTLIST method name CDATA #REQUIRED>
<!ATTLIST method code CDATA #REQUIRED>
<!ATTLIST method value CDATA #REQUIRED>
```

An example XML file of the results of method invocations.

```
<!DOCTYPE resource SYSTEM "invokedMethodsResult.dtd">

<invokedMethodsResult>
  <method name="setTemperature" code="0" value=""/>
  <method name="getTemperature" code="0" value="21.3"/>
</invokedMethodsResult>
```

APPENDIX E

Workflow DTD

The DTD of a workflow definition.

```
<!-- Goal ------------------------------------------------->
<!ELEMENT goal (incentive,
                goal*,
                activity*)>
<!ATTLIST goal id CDATA>
<!ATTLIST goal type CDATA>

<!-- Incentive -------------------------------------------->
<!ELEMENT incentive (isDefined*,
                     setDefined*,
                     condition*,
                     compare*,
                     findResource*,
                     useResource*,
                     execute*,
                     wait*)>
<!ATTLIST incentive interval CDATA "10000">
<!ATTLIST incentive iftrue CDATA>
<!ATTLIST incentive iffalse CDATA>
<!ATTLIST incentive binding (and|or) "and">

<!-- Activity --------------------------------------------->
<!ELEMENT activity (isDefined*,
                    setDefined*,
                    condition*,
                    compare*,
                    findResource*,
                    useResource*,
                    execute*,
                        wait*)>
<!ATTLIST activity id CDATA>
<!ATTLIST activity type CDATA>

<!-- IsDefined ------------------------------------------->
<!ELEMENT isDefined EMPTY>
<!ATTLIST isDefined name CDATA #REQUIRED>

<!-- SetDefined ------------------------------------------>
<!ELEMENT setDefined EMPTY>
<!ATTLIST setDefined name CDATA #REQUIRED>
<!ATTLIST setDefined value CDATA #REQUIRED>

<!-- Condition ------------------------------------------->
<!ELEMENT condition (isDefined*,
                     setDefined*,
                     condition*,
                     compare*,
                     findResource*,
                     useResource*,
                     execute*,
                     wait*)>
<!ATTLIST condition iftrue CDATA>
<!ATTLIST condition iffalse CDATA>
<!ATTLIST condition binding (and|or) "and">

<!-- Compare --------------------------------------------->
<!ELEMENT compare EMPTY>
<!ATTLIST compare ls CDATA #REQUIRED>
<!ATTLIST compare rs CDATA #REQUIRED>
<!ATTLIST compare type (StringEqualsIgnoreCase|
                        StringEquals|
                        NumericEquals|
                        NumericGreater|
                        NumericGreaterEquals|
                        NumericLess|
                        NumericLessEquals|
                        Boolean) #REQUIRED>


<!-- FindResource ---------------------------------------->
<!ELEMENT findResource EMPTY>
<!ATTLIST findResource type CDATA #REQUIRED>
<!ATTLIST findResource location CDATA>
<!ATTLIST findResource name CDATA>
<!ATTLIST findResource local (true|false) "false">

<!-- UseResource ----------------------------------------->
<!ELEMENT useResource (invoke*)>
<!ATTLIST useResource name CDATA #REQUIRED>
```

```

<!-- Invoke ---------------------------------------------->
<!ELEMENT invoke (parameter*)>
<!ATTLIST invoke method CDATA #REQUIRED>
<!ATTLIST invoke name CDATA>

<!-- Parameter ------------------------------------------->
<!ELEMENT parameter EMPTY>
<!ATTLIST parameter value CDATA #REQUIRED>

<!-- Execute --------------------------------------------->
<!ELEMENT execute EMPTY>
<!ATTLIST execute id CDATA>
<!ATTLIST execute control CDATA>

<!-- Wait ------------------------------------------------>
<!ELEMENT wait EMPTY>
<!ATTLIST wait milliseconds CDATA #REQUIRED>
```

APPENDIX F

Workflow XML example

An example XML file of a workflow definition. This is a complete example of controlling the temperature of the room the client currently is in. The temperature is controlled to match the user defined variable *desiredTemperature*.

A walk-through of the workflow is provided as comments in the xml example.

```
<!DOCTYPE resource SYSTEM "workflow.dtd">

<!-- Top level goal with the ontology individual for
     controlling temperature set. -->
<goal id="controlTemperature" type="Control_temperature">

  <!-- This is the access point of the worfklow. It is defined
       with check intervals of 10 sec, execute the activity
       with id 'control' when it turns true, and to nothing
       when it becomes false. When the binding is set to "and"
       it will only execute the next statment if the prior
       returned true. Likewise, if it is "or" it will drop the
       rest of the statments if a statement return true. -->
  <incentive interval="10000" iftrue="control"
             iffalse="" binding="and">
    <!-- Check if the user has defined a desired temperature. -->
    <isDefined name="desiredTemperature"/>
    <!-- Look for a location sensor running locally on the
         client and store it in a variable called locationSensor
         if found. -->
    <findResource type="Location_sensor"
                  name="locationSensor"
                  local="true"/>
    <!-- Executes the activity with id "getLocation" -->
    <execute id="getLocation"/>
  <!-- Look for a temperature sensor in the location specified
       by the "currentLocation" variable and store it in a
       variable called temperatureSensor, if found. -->
  <findResource type="Temperature_sensor"
                location="${currentLocation}"
                name="temperatureSensor"
                local="false"/>
  <!-- Look for a temperature actuator in the location
       specified by the "currentLocation" variable and
       store it in a variable called temperatureSensor,
       if found. -->
  <findResource type="Temperature_actuator"
                location="${currentLocation}"
                name="temperatureActuator"
                local="false"/>
</incentive>

<!-- An activity that gets the current location of the
     client. -->
<activity id="getLocation">
  <!-- Use the resource in variable "locationSensor" -->
  <useResource name="locationSensor">
    <!-- Invoke the method "getLocation" and store the result
         in the variable "currentLocation". -->
    <invoke method="getLocation" name="currentLocation"/>
  </useResource>
</activity>

<!-- The main activity for this goal. It has an ontology type
     of adjust temperature. -->
<activity id="control" type="Adjust_temperature">
  <!-- Execute the "readTemperature" activity. -->
  <execute id="readTemperature"/>
  <!-- Perform a condition. If true, execute "turnOnActuator"
       activity, if false, execute "turnOffActuator". -->
  <condition iftrue="turnOnActuator"
             iffalse="turnOffActuator"
             binding="and">
    <!-- Compare the values of the two variables
         "currentTemperature" and "desiredTemperature". -->
    <compare ls="${currentTemperature}"
             rs="${desiredTemperature}"
             type="NumericLess"/>
  </condition>
  <!-- Wait 10 seconds. -->
  <wait milliseconds="10000"/>
  <!-- Execute itself again. (Loop) -->
  <execute id="control"/>
</activity>

```

```
  <activity id="turnOffActuator">
    <!-- Sets the variable "level" to the value "0". -->
    <setDefined name="level" value="0"/>
    <!-- Executes "setTemperatureActuator" activity. -->
    <execute id="setTemperatureActuator"/>
  </activity>

  <activity id="turnOnActuator">
    <!-- Sets the variable "level" to the value "1". -->
    <setDefined name="level" value="1"/>
    <!-- Executes "setTemperatureActuator" activity. -->
    <execute id="setTemperatureActuator"/>
  </activity>

  <activity id="readTemperature">
    <!-- Use the resource stored in variable
         "temperatureSensor". -->
    <useResource name="temperatureSensor">
      <!-- Invoke the method "getTemperature" and store the
           result in the variable "currentTemperature". -->
      <invoke method="getTemperature" name="currentTemperature"/>
    </useResource>
  </activity>

  <activity id="setTemperatureActuator">
    <!-- Use the resource stored in variable
         "temperatureActuator". -->
    <useResource name="temperatureActuator">
      <!-- Invoke the method "setLevel" with a parameter. -->
      <invoke method="setLevel">
        <!-- Set the parameter to the value of the variable
             "level". -->
        <parameter value="${level}"/>
      </invoke>
    </useResource>
  </activity>

</goal>
```

APPENDIX G

Workflow server and connector DTD

The DTD for the status message passed between workflows and an example xml message.

```
<!ELEMENT workflowStatus (participants,log)>

<!ELEMENT participants (participant*)>
<!ELEMENT participant EMPTY>

<!ELEMENT log (entry*)>
<!ELEMENT entry (#PCDATA)>

<!ATTLIST workflowStatus running (true|false) #REQUIRED>

<!ATTLIST participant peerName CDATA #REQUIRED>
<!ATTLIST participant peerID CDATA #REQUIRED>

<!ATTLIST log code CDATA #REQUIRED>

<!ATTLIST entry activity CDATA #REQUIRED>
<!ATTLIST entry statement CDATA #REQUIRED>
<!ATTLIST entry level CDATA #REQUIRED>
```

The xml example of a status message sent from the RemoteWorkflowServer component to the RemoteWorkflowConnector component.

```xml
<!DOCTYPE resource SYSTEM "workflowStatus.dtd">

<workflowStatus running="true">

  <participants>
    <participant peerName="Christian"
                 peerID="uuid-59616261646162614E504720503250...
                         3317FEC6556FD04675B397F7AE372BEC9403"/>
    <participant peerName="Kjell"
                 peerID="uuid-92873926536162614E504720503250...
                         3317FEC6556FD04675B397F7AE3E7BA29E5F"/>
  </participants>

  <log code="27">
    <entry activity="activity-1894712674"
           statement="UseResource"
           level="1">
      Method invoked successfully. Result: 19.34
    </entry>
  </log>

</workflowStatus>
```

APPENDIX H

Arbitration DTD

The DTD of a arbitration message.

```
<!ELEMENT arbitration (conflict?,solution)>
<!ATTLIST arbitration peerName CDATA #REQUIRED>
<!ATTLIST arbitration peerID CDATA #REQUIRED>

<!ELEMENT confict EMPTY>
<!ATTLIST confict case1 CDATA #REQUIRED>
<!ATTLIST confict case2 CDATA #REQUIRED>
<!ATTLIST confict location CDATA #REQUIRED>
<!ATTLIST confict participants CDATA #REQUIRED>
<!ATTLIST confict solution CDATA #REQUIRED>
<!ATTLIST confict parameter CDATA #REQUIRED>

<!ELEMENT solution (parameter*)>
<!ATTLIST solution type (wait|
                         waitRemote|
                         halt|
                         haltRemote|
                         adapt|
                         adaptRemote) #REQUIRED>

<!ELEMENT parameter EMPTY>
<!ATTLIST parameter name CDATA #REQUIRED>
<!ATTLIST parameter value CDATA #REQUIRED>
```

An example of an arbitration xml message between two clients. This example shows the adaption strategy *mean* for the workflow xml example in Appendix F, where the variable *desiredTemperature* is passed to the other client to create a new common mean value.

```
<!DOCTYPE resource SYSTEM "arbitration.dtd">

<arbitration peerName="Chris"
             peerID="uuid-59616261646162614E504720503250...
                     3317FEC6556FD04675B397F7AE372BEC9403">

  <confict case1="Control_light" case2="Control_light"
           location="1.0" participants="0.0"
           solution="Use_case_2" parameter="null"/>

  <solution type="adaptRemote">
    <parameter name="workflowID" value="952836725"/>
    <parameter name="pipeID"
               value="uuid-E62C28E82FA6162614E504720503250...
                      3317FEC6556FD04675B397F7AE372BEC9403"/>
    <parameter name="desiredTemperature" value="21.5"/>
  </solution>

</arbitration>
```

APPENDIX I

Workflow description DTD

The DTD of workflow description message used by Scheduler and ConflictDiscoverer.

```
<!ELEMENT workflows (goal*,activity*)>
<!ATTLIST workflows peerName CDATA #REQUIRED>
<!ATTLIST workflows peerID CDATA #REQUIRED>

<!ELEMENT goal (goal*,activity*)>
<!ATTLIST goal id CDATA #REQUIRED>
<!ATTLIST goal type CDATA #REQUIRED>
<!ATTLIST goal location CDATA #REQUIRED>
<!ATTLIST goal pipeID CDATA #REQUIRED>
<!ATTLIST goal workflowID CDATA #REQUIRED>

<!ELEMENT activity EMPTY>
<!ATTLIST activity id CDATA #REQUIRED>
<!ATTLIST activity type CDATA #REQUIRED>
<!ATTLIST activity location CDATA #REQUIRED>
<!ATTLIST activity pipeID CDATA #REQUIRED>
<!ATTLIST activity workflowID CDATA #REQUIRED>
```

An example of a workflow description file generated by the Scheduler and parsed by the ConflictDiscoverer to check for possible conflicts.

```
<!DOCTYPE resource SYSTEM "workflowDescription.dtd">

<workflows peerName="Kjell"
           peerID="uuid-29616261646162614E504720503257...
                   3317FEC6556FD04675B397F7AE372BEC9403">

  <goal id="controlTemperature"
        type="Control_temperature"
        location="White_room"
        pipeID="uuid-A3916261646162614E504720503229...
                3317FEC6556FD04675B397F7AE372BECAC24"
        workflowID="2937261923">

    <activity id="control"
              type="Adjust_temperature"
              location="White_room"
              pipeID="uuid-59616261646162614E504720503253...
                      3317FEC6556FD04675B397F7AE372BECDA91"
              workflowID="2937261923">

  </goal>

  <activity id="activity-29438727"
            type="Adjust_light"
            location="Green_room"
            pipeID="uuid-E928A826EC6162614E504720503250...
                    E27CFEC6556FD04675B397F7AE372BEC2F35"
            workflowID="182749012">

</workflows>
```

Glossary

Actuator	A person or machine that can do work, move, or act.
AD	Architectural Description
Ad hoc	An ad hoc arrangement is one which is unplanned and which takes place only because a situation has made it necessary.
AI	Artificial Intelligence
Ambient computing	Is a combination of a number of paradigms; ubiquitous computing, pervasive computing, and artificial intelligence.
Arbitration	Arbitration is the judging of a dispute.
Architecture	The manner in which the components of a computer or computer system are organized and integrated.
Asynchronous	Not synchronous. See *Synchronous*
BDI	Belief-Desire-Intention
CAWES	Context-Aware Workflow Enactment Service
CBR	Case-based reasoning
CC/PP	Composite Capability/Preference Profile
CDP	Cooperative Distributed Planning
CHAT	Cultural Historical Activity Theory
CoBrA	Context Broker Architecture
COLIBRI	Cases and Ontology Libraries Integration for Building Reasoning Infrastructures
COWCS	Cooperative Workflow Coordination Service
CSCW	Computer Supported Cooperative Work
Cyber foraging	Utilize compute resources available in the environment to augment the capabilities of mobile devices.
DAG	Directed, Acyclic Graph
DCP	Distributed, Continual Planning

DL	Description Logics
DTD	Document Type Definition
Fuzzy	Lacking in clarity or definition.
GUI	Graphic User Interface
IrDA	Infrared Data Association
J2ME	Java 2 Micro Edition
J2SE	Java 2 Standard Edition
JCAF	Java Context-Aware Framework
jCOLIBRI	Java Cases and Ontology Libraries Integration for Building Reasoning Infrastructures
KBS	Knowledge Based System
KI-CBR	Knowledge Intensive Cased-Based Reasoning
LORA	LOgic of Rational Agents
MANet	Mobile Ad-hoc Network
MAS	Multiagent systems
MDP	Markov Decision Process
MOWAHS	MObile Work Across Heterogeneous Systems
NAT	Network Address Translation
NDP	Negotiated Distributed Planning
OWL	Ontology Web Language
P2P	See *Peer-to-peer*
PDA	Personal Digial Assistant
Peer-to-peer	A peer-to-peer (or P2P) computer network is a network that relies primarily on the computing power and bandwidth of the participants in the network rather than concentrating it in a relatively low number of servers. P2P networks are typically used for connecting nodes via largely ad hoc connections.

Pervasive computing	Seamlessly integrates computation into the environment, rather than having computers which are distinct objects.
PSM	Problem Solving Method
RAD	Rapid Application Development
RDF	Resource Description Framework
Reasoning	Reasoning is the process by which you reach a conclusion after considering all the facts.
RMI	Remote Method Invocation
SCBR	Soft Case-Based Reasoning
Scheduling	Scheduling is the process of making a plan that gives a list of goals and activities, together with the times at which each goal or activity should be executed.
Sensor	A device that measures or detects a real-world condition, such as motion, heat or light and converts the condition into an analogue or digital representation. An optical sensor detects the intensity or brightness of light, or the intensity of red, green and blue for colour systems.
SHA-1	Secure Hash Algorithm - 1
Situated	To be placed in a site, situation, context, or category.
Smart spaces	Ordinary environments equipped with sensing systems that can perceive and react to people without requiring them to wear any special equipment.
SOC	Service-oriented computing
SoC	Separation of Concern
SOCAM	Service-oriented Context-Aware Middleware
SWP	Smart Work Processes
Synchronous	Occurring, existing, or operating at the same time.
Ubiquitous computing	See *Pervasive computing*
URI	Universal Resource Identifier
URN	Universal Resource Name
W3C	World Wide Web Consortium
WfMC	Workflow Management Coalition
XML	Extended Markup Language

Bibliography

[1] AAAI. Planning and Scheduling. http://www.aaai.org/AITopics/html/planning.html, Accessed may 25th 2007 2003.

[2] Agnar Aamodt. Knowledge-intensive case-based reasoning in creek. In *ECCBR*, pages 1–15. Springer Berlin / Heidelberg, 2004.

[3] Agnar Aamodt and Enric Plaza. Case-Based Reasoning: Foundational Issues, Methodological Variations, and System Approaches. *Artificial Intelligence Communications*, 7(1):39–52, 1994.

[4] Ananda Amatya. Requirements. http://www.dcs.warwick.ac.uk/ doron/-course/cs223/chap4.ppt, Accessed december 10th 2006.

[5] AmbieSense. AmbieSense EU IST project. http://www.ambiesense.net/, Accessed june 12th 2007 2005.

[6] J. A. Ambros-Ingerson and S. Steel. Integrating planning, execution and monitoring. In J. Allen, J. Hendler, and A. Tate, editors, *Readings in Planning*, pages 735–740. Kaufmann, San Mateo, CA, 1990.

[7] Rajesh Balan, Jason Flinn, M. Satyanarayanan, Shafeeq Sinnamohideen, and Hen-I Yang. The case for cyber foraging. In *EW10: Proceedings of the 10th workshop on ACM SIGOPS European workshop: beyond the PC*, pages 87–92, New York, NY, USA, 2002. ACM Press.

[8] Rajesh Balan, Jason Flinn, M. Satyanarayanan, Shafeeq Sinnamohideen, and Hen-I Yang. The case for cyber foraging. In *EW10: Proceedings of the 10th workshop on ACM SIGOPS European workshop: beyond the PC*, pages 87–92, New York, NY, USA, 2002. ACM Press.

[9] Matthias Baldauf, Schahram Dustdar, and Florian Rosenberg. A Survey on Context-Aware Systems. *International Journal of Ad Hoc and Ubiquitous Computing*, 2004.

[10] Matthias Baldauf, Schahram Dustdar, and Florian Rosenberg. A Survey on Context-Aware Systems. *International Journal of Ad Hoc and Ubiquitous Computing*, 2004.

[11] J. F. Baldwin, editor. *Fuzzy Logic*.

[12] Jakob Bardram. Plans as situated action: An activity theory approach to workflow systems. In *ECSCW*, pages 17–, 1997.

[13] Jakob Bardram. Plans as situated action: An activity theory approach to workflow systems. In *ECSCW*, pages 17–, 1997.

[14] Jakob E. Bardram. The java context awareness framework (jcaf) - a service infrastructure and programming framework for context-aware applications. In *Proceedings of the 3rd International Conference on Pervasive Computing (Pervasive 2005), Lecture Notes in Computer Science*, pages 98–115. Springer Berlin / Heidelberg, 2005.

[15] Allen Dutoit Bernd Bruegge. Requirements Elicitation. http://www.cs.fsu.edu/ gaitrosd/classes/cop3331/Chapter4/ch4lect1.ppt, Accessed december 10th 2006.

[16] Tim Berners-Lee, James Hendler, and Ora Lassila. The Semantic Web. *Scientific American*, 2001.

[17] Craig Boutilier, Thomas Dean, and Steve Hanks. Decision-theoretic planning: Structural assumptions and computational leverage. *Journal of Artificial Intelligence Research*, 11:1–94, 1999.

[18] Michael Brenner and Bernhard Nebel. Continual planning and acting in dynamic multiagent environments. In *PCAR '06: Proceedings of the 2006 international symposium on Practical cognitive agents and robots*, pages 15–26, New York, NY, USA, 2006. ACM Press.

[19] Barry Brown and Nicola Green, editors. *Wireless world: social and interactional aspects of the mobile age*. Springer-Verlag New York, Inc., New York, NY, USA, 2002.

[20] P. Brzillon and J. Pomerol. Contextual knowledge sharing and cooperation in intelligent assistant systems. volume 62 (3) of *Le Travail Humain*, pages 223–246, 1999.

[21] National Research Council Canada. FuzzyJess. http://www.nrc-cnrc.gc.ca/, Accessed October 17th 2006.

[22] Jörg Cassens and Anders Kofod-Petersen. Using Activity Theory to Model Context Awareness: a Qualitative Case Study. In Geoff C. J. Sutcliffe and Randy G. Goebel, editors, *Proceedings of the Nineteenth International Florida Artificial Intelligence Research Society Conference*, pages 619–624, Melbourne Beach, 2006. AAAI Press.

[23] Marcelo Cataldo, Patrick A. Wagstrom, James D. Herbsleb, and Kathleen M. Carley. Identification of coordination requirements: implications for the design of collaboration and awareness tools. In *CSCW '06: Proceedings of the 2006 20th anniversary conference on Computer supported cooperative work*, pages 353–362, New York, NY, USA, 2006. ACM Press.

[24] European Usability Support Center. Requirements Engineering and Specification in Telematics, Scenario Building. http://www.ucc.ie/hfrg/projects/respect/urmethods/scenario.htm, Accessed december 10th 2006.

[25] Guanling Chen and David Kotz. A survey of context-aware mobile computing research. Technical report, Dept. of Computer Science, Dartmouth College, November 2000.

[26] Harry Chen. *An Intelligent Broker Architecture for Pervasive Context-Aware Systems*. PhD thesis, University of Maryland, Baltimore County, December 2004.

[27] Bradley J. Clement and Edmund H. Durfee. Top-down search for coordinating the hierarchical plans of multiple agents. In *AGENTS '99: Proceedings of the third annual conference on Autonomous Agents*, pages 252–259, New York, NY, USA, 1999. ACM Press.

[28] Workflow Management Coalition. Terminology & glossary. Technical Report 3, Workflow Management Coalition, 1999.

[29] Philip R. Cohen and Hector J. Levesque. Intention is choice with commitment. *Artificial intelligence*, 42(2-3):213–261, 1990.

[30] CollabNet. JXTA. www.jxta.org, Accessed feb 12th 2007 2006.

[31] Bel&#233; D&#237;az-Agudo and Pedro A. Gonz&#225;lez-Calero. Cbronto: A task/method ontology for cbr. In *Proceedings of the Fifteenth International Florida Artificial Intelligence Research Society Conference*, pages 101–105. AAAI Press, 2002.

[32] Marie desJardins, Edmund H. Durfee, Charles L. Ortiz Jr., and Michael Wolverton. A survey of research in distributed, continual planning. *AI Magazine*, 20(4):13–22, 1999.

[33] Anind K. Dey. Understanding and Using Context. *Personal and Ubiquitous Computing Journal*, 5:4–7, 2001.

[34] Anind K. Dey and Gregory D. Abowd. Towards a better understanding of context and context-awareness. *the 1st International Symposium on Handheld and Ubiquitous Computing*, pages 304–307, 1999.

[35] Anind K. Dey, Daniel Salber, and Gregory D. Abowd. A Conceptual Framework and a Toolkit for Supporting the Rapid Prototyping of Context-Aware Applications. *Human-Computer Interaction (HCI) Journal*, 16:97–166, 2001.

[36] Richard J. Doyle, David Atkinson, and Rajkumar Doshi. Generating perception requests and expectations to verify the execution of plans. In *AAAI*, pages 81–88, 1986.

[37] Norfolk East of England RDSU and Suffolk (Norwich). Epidemiological Research Strategies & Study Design. http://www.east-of-england-rdsu.org.uk/resources/docs/infosheet-2.pdf, Accessed december 10th 2006.

[38] Jacques Ferber. *Multi-Agent Systems: An Introduction to Distributed Artificial Intelligence*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1999.

[39] Mikael Kirkeby Fidjeland. Distributed knowledge in case-based reasoning - knowledge sharing and reuse within the semantic web. Master's thesis, Norwegian University of Science and Technology, Department of Computer and Information Science, 2006.

[40] Geraldine Fitzpatrick.

[41] J. Flinn, D. Narayanan, and M. Satyanarayanan. *Self-tuned remote execution for pervasive computing*, pages 61–66. HotOS-VIII. 2001.

[42] Jason Flinn, SoYoung Park, and M. Satyanarayanan. Balancing performance, energy, and quality in pervasive computing. In *ICDCS '02: Proceedings of the 22 nd International Conference on Distributed Computing Systems (ICDCS'02)*, page 217, Washington, DC, USA, 2002. IEEE Computer Society.

[43] James Frye, Rajagopal Ahnanthanarayanan, and Dahrmendra S. Modha. Towards real-time, mouse scale cortical simulations. Technical report, IBM, 2007.

[44] Ayse Göker and Hans I. Myrhaug. User context and personalisation. In *ECCBR Workshops*, pages 1–7, 2002.

[45] Sachin Goyal and John Carter. A Lightweight Secure Cyber Foraging Infrastructure for Resource-Constrained Devices.

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

[47] IEEE Architecture Working Group. Ieee std 1471-2000, recommended practice for architectural description of software-intensive systems. Technical report, IEEE, 2000.

[48] Frode Hauso and Øivind Røed. Adaptive Mobile Work Processes. Technical report, 2005.

[49] James Hendler, Austin Tate, and Mark Drummond. Ai planning: systems and techniques. *AI Mag.*, 11(2):61–77, 1990.

[50] Karen Henricksen, Jadwiga Indulska, and Andry Rakotonirainy. Modeling context information in pervasive computing systems. In *Pervasive '02: Proceedings of the First International Conference on Pervasive Computing*, pages 167–180, London, UK, 2002. Springer-Verlag.

[51] Michael N. Huhns, Munindar P. Singh, Mark Burstein, Keith Decker, Ed Durfee, Tim Finin, Les Gasser, Hrishikesh Goradia, Nick Jennings, Kiran Lakkaraju, Hideyuki Nakashima, Van Parunak, Jeffrey S. Rosenschein, Alicia Ruvinsky, Gita Sukthankar, Samarth Swarup, Katia Sycara, Milind Tambe, Tom Wagner, and Laura Zavala. Research directions for service-oriented multiagent systems. *IEEE Internet Computing*, 9(6):65–70, 2005.

[52] Robert Johansen. Groupware. Computer Support for Business Teams. *The Free Press, New York and London*, 1988.

[53] Pål Johan Karlsen. *Slik får du bedre hukommelse*. H. Aschehoug & Co., 2004.

[54] Anders Kofod-Petersen. *A Case-Based Approach to Realising Ambient Intelligence among Agents*. PhD thesis, Norwegian University for Science and Technology, 2007.

[55] Anders Kofod-Petersen and Agnar Aamodt. Case-Based Situation Assessment in a Mobile Context-Aware System. In Antonio Krüger and Rainer Malaka, editors, *Artificial Intelligence in Mobile Systems 2003 (AIMS)*, pages 41–49. Universität des Saarlandes, October 2003.

[56] Anders Kofod-Petersen and Marius Mikalsen. Context: Representation and Reasoning – Representing and Reasoning about Context in a Mobile Environment. *Revue d'Intelligence Artificielle*, 19(3):479–498, 2005.

[57] Janet L. Kolodner. *Case-Based Reasoning*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1993.

[58] Mari Korkea-aho. Context-Aware Applications Survey. http://users.tkk.fi/ mkorkeaa/doc/context-aware.html, Accessed june 8th 2000.

[59] Reading Success Lab. Descriptive Research - Defined. http://www.cognitive-aptitude-assessment-software.com/Glossary/DescriptiveResearch.html, Accessed december 10th 2006.

[60] Danny B. Lange and Mitsuru Oshima. *Programming and deploying Java mobile agents with aglets*. Addison Wesley, 1998.

[61] Bruno Latour. *Science in Action: How to Follow Scientists and Engineers Through Society*. Harvard University Press, October 1988.

[62] David Leake. CBR in context: The present and future. Technical report, 1996.

[63] David B. Leake. *Case-Based Reasoning: Experiences, Lessons and Future Directions*. MIT Press, Cambridge, MA, USA, 1996.

[64] Kalle Lyytinen and Youngjin Yoo. Introduction. *Commun. ACM*, 45(12):62–65, 2002.

[65] Nico Maibaum and Thomas Mundt. Jxta: A technology facilitating mobile peer-to-peer networks. In *MOBIWAC '02: Proceedings of the International Workshop on Mobility and Wireless Access*, page 7, Washington, DC, USA, 2002. IEEE Computer Society.

[66] Jon Ole Nødtvedt and Man Hoang Nguyen. Mobility and context-awareness in workflow systems. Technical report, Norwegian University for Science and Technology, 2004.

[67] Nils J. Nilsson. *Artificial Intelligence: A New Synthesis*. Morgan Kaufmann, 1998.

[68] Natalya F. Noy and Deborah L. McGuinness. Ontology development 101: A guide to creating your first ontology. Technical report, Stanford University School of Medicine, 2001.

[69] Georgia Tech College of Computing. Requirements Elicitation. http://www-static.cc.gatech.edu/classes/AY2002/cs6300_fall/req1/Reqts.ppt, Accessed december 10th, 2006 2000.

[70] Purdue University OWL. Types of Research Papers. http://owl.english.purdue.edu/workshops/hypertext/ResearchW/types.html, Accessed december 10th 2006.

[71] Mr. Jason Pascoe. Adding generic contextual capabilities to wearable computers. In *ISWC '98: Proceedings of the 2nd IEEE International Symposium on Wearable Computers*, page 92, Washington, DC, USA, 1998. IEEE Computer Society.

[72] Sobah Abbas Petersen and Anders Kofod-Petersen. The non-accidental tourist: Using ambient intelligence for enhancing tourist experiences. In Luis Camarinha-Matos, Hamideh Afsarmanesh, and Martin Ollus, editors, *Proceedings of the 7th IFIP Working Conference on Virtual Enterprises*, volume 224 of *IFIP International Federation for Information Processing*, Helsinki, Finland, September 2006. Springer Verlag.

[73] NASA Ames research center. Planning and Scheduling. http://ic.arc.nasa.gov/projects/remote-agent/pstext.html, Accessed may 25th 2007 2003.

[74] USA Richard M. Jacobs, Villanova Univeristy. Educational Research: Descriptive Research. http://www83.homepage.villanova.edu/richard.jacobs/EDUptive.ppt, Accessed december 10th 2006.

[75] Kjell Martin Rud and Christian Indahl. Architectural approaches supporting scheduling and arbitration of workflow processes in a cooperative environment. Technical report, Norwegian University for Science and Technology, 2006.

[76] Stuart J. Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Pearson Education, 2003.

[77] M. Satyanarayanan. Pervasive computing: vision and challenges. *Personal Communications, IEEE*, 8(4):10–17, 2001.

[78] Roger C. Schank. *Dynamic Memory: A Theory of Reminding and Learning in Computers and People*. Cambridge University Press, New York, NY, USA, 1983.

[79] Roger C. Schank. Goal-based scenarios: Case-based reasoning meets learning by doing. In David Leake, editor, *Case-Based Reasoning: Experiences, Lessons & Future Directions*, pages 295–347. AAAI Press/The MIT, 1996.

[80] Roger C. Schank and Robert P. Abelson. Knowledge and memory: The real story. pages 1–85, 1995.

[81] Bill Schilit, Norman Adams, and Roy Want. Context-aware computing applications. In *IEEE Workshop on Mobile Computing Systems and Applications*, Santa Cruz, CA, US, 1994.

[82] Bill Schilit and M. Theimer. Disseminating active map information to mobile hosts. *IEEE Network*, 8(5):22–32, 1994.

[83] Albrecht Schmidt, Kofi Asante Aidoo, Antti Takaluoma, Urpo Tuomela, Kristof Van Laerhoven, and Walter Van de Velde. Advanced interaction in context. In *HUC '99: Proceedings of the 1st international symposium on Handheld and Ubiquitous Computing*, pages 89–101, London, UK, 1999. Springer-Verlag.

[84] M. J. Schoppers. Universal plans for reactive robots in unpredictable environments. In John McDermott, editor, *Proceedings of the Tenth International Joint Conference on Artificial Intelligence (IJCAI-87)*, pages 1039–1046, Milan, Italy, 1987. Morgan Kaufmann publishers Inc.: San Mateo, CA, USA.

[85] Guus Schreiber, Bob Wielinga, Robert de Hoog, Hans Akkermans, and Walter Van de Velde. Commonkads: A comprehensive methodology for kbs development. *IEEE Expert: Intelligent Systems and Their Applications*, 09(6):28–37, 1994.

[86] N. Shadbolt, W. Hall, and T. Berners-Lee. The semantic web revisited. In *IEEE Intelligent Systems*, pages 96–101, 2006.

[87] Simon C. K. Shiu and Sankar K. Pal. *Foundations of soft case-based reasoning*. Wiley, 2004.

[88] Carl-Fredrik Sørensen. *Adaptive Mobile Work Processes in Context-Rich, Heterogeneous Environments*. PhD thesis, Norwegian University for Science and Technology, 2005.

[89] Carl-Fredrik Sørensen, Alf Inge Wang, and Reidar Conradi. Support of smart work processes in context rich environments. In *In Proceedings of the IFIP TC8 Working Conference on Mobile Information Systems - 2005*, Leeds, UK, 2005.

[90] Peter Stone and Manuela Veloso. Task decomposition, dynamic role assignment, and low-bandwidth communication for real-time strategic teamwork. *Artificial Intelligence*, 1999.

[91] Lucy Suchman. *Plans and situated actions. The problem of human-machine communication*. Cambridge University Press, 1987.

[92] Lucy A. Suchman. *Plans and situated actions: the problem of human-machine communication*. Cambridge University Press, New York, NY, USA, 1987.

[93] Richard S. Sutton and Andrew G. Barto. *Reinforcement learning, an introduction*. MIT Press, 2000.

[94] M. Tambe. Towards flexible teamwork. *Journal of Artificial Intelligence Research*, 7:83–124, 1997.

[95] Gu Tao, Pung Hung Keng, and Zhang Da. SOCAM. http://www.comp.nus.edu.sg/ gutao/gutao_NUS/SOCAM.htm, Accessed december 4nd 2005.

[96] Various Encoder: the newsletter for seattle robotics society. Fuzzy Logic - An Introduction. http://www.seattlerobotics.org/encoder/mar98/fuz/fl_part1.html, Accessed September 25th 2006.

[97] Lev Semenovic Vygotskij. *Mind in society : the development of higher psychological processes*. Cambridge, Mass. : Harvard University Press, 1978.

[98] Alf Inge Wang. MOWAHS. http://www.mowahs.com/, Accessed September 19th 2006.

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

[100] Mark Weiser and John Seely Brown. The coming age of calm technolgy. pages 75–85, 1997.

[101] Gerhard Weiss, editor. *Multiagent systems: a modern approach to distributed artificial intelligence*. MIT Press, Cambridge, MA, USA, 1999.

[102] Matthias Wieland, Oliver Kopp, Daniela Nicklas, and Frank Leymann. Towards context-aware workflows. In *Proceedings of the Ubiquitous Mobile Information and Collaboration Systems (Caise'07 Workshop), Trondheim, Norway, June 11-12th, 2007*, pages 0–0. University of Stuttgart : Collaborative Research Center SFB 627 (Nexus: World Models for Mobile Context-Based Systems), Springer Verlag, June 2007.

[103] Various Wikipedia.org. Artificial neural networks. http://en.wikipedia.org/wiki/Artificial_neural_networks, Accessed may 14th 2007.

[104] Various Wikipedia.org. Bayesian network. http://en.wikipedia.org/wiki/Bayesian_network, Accessed March 29 2007.

[105] Various Wikipedia.org. Case-based reasoning. http://en.wikipedia.org/wiki/Case-based_reasoning, Accessed may 12th 2007.

[106] Various Wikipedia.org. JXTA. http://en.wikipedia.org/wiki/JXTA, Accessed January 29 2007.

[107] Various Wikipedia.org. memory. http://en.wikipedia.org/wiki/Memory, Accessed March 8 2007.

[108] Various Wikipedia.org. Mobile ad-hoc network. http://en.wikipedia.org/wiki/Mobile_ad-hoc_network, Accessed may 21th 2007.

[109] Michael Wooldridge. *Reasoning about rational agents*. MIT Press, 2000.