An Aspect-Oriented Approach to Adaptive Systems

Master thesis by
John Christian Hveding

Abstract

Adaptive systems are systems that react to changes in their environment and adapt to these changes by changing their behavior. The FAMOUS project aims to build an adaptive system by creating a generic middleware platform. This project explores how adaptive systems in general and the FAMOUS project in particular can benefit from using aspect-oriented technology. We propose using run-time aspect weaving to perform adaptations. We create a prototype to demonstrate how one can model aspects for adaptations. We suggest that variability engineering of the applications for an adaptive platform can benefit from aspect-oriented software development.

Acknowledgements

I would like to thank SINTEF ICT and Svein Olav Hallsteinsen and Erlend Stav in particular. They have been more than patient with me, and have answered my not always intelligent questions and given feedback on my incomprehensible drafts.

I would like to thank Alf Inge Wang for providing valuable feedback throughout the project period and for intensive proof reading in the later stages of the project.

I would like to thank my girlfriend, Cathrine Williamsen for tolerating my many late nights and erratic work schedule. I would also like to thank her for putting up with my mood swings and for helping motivate me when things were not going my way.

I would like to thank everybody at the Department of Computer and Information Science and everybody else which contributed to having a happy six years at NTNU.

John Christian Hveding

Table of Contents

Figure List

Table List

1 Introduction

The pervasiveness of handheld computer devices has given rise to a new set of challenges for software engineers. Users of handheld devices operate in varying conditions, and are exposed to unstable networks, variable lighting conditions, noisy situations or being engaged in performing activities while using the device. One way to improve the devices' usability and usefulness to the user is to take the users' environment into account and adapt the application to the user's context.

The FAMOUS project [1] aims at providing support for developing such adaptive systems by creating a generic engineering framework for adaptive mobile services. This framework will include support for standard mechanisms needed by adaptive applications; context models that allow applications to access, update and reason about contextual information and mechanisms for run-time adaptation.

To achieve this, the framework is currently based on component frameworks, variability engineering, property modeling, architectural reflection and generic middleware to handle adaptation [2]. An application developed for the FAMOUS platform is developed as a component framework. A model of the application is kept by middleware at all times. When changes occur in the context, requiring adaptation, the middleware analyze the meta-model to find the configuration of the components that best fits the current context. The application is then reconfigured to meet the new context requirements.

The FAMOUS project also has as an objective to "develop and validate software engineering solutions for the construction of mobile adaptive services" [3]; hereunder exploring new software engineering methods to see if they can be used to develop adaptive systems more effectively.

The MADAM project aims to provide a generic middleware platform to monitor context and make decisions based on context. "The overall objective of MADAM is to provide software engineers with suitable means to develop mobile adaptive applications. The project will investigate the idea that adaptivity can be supported by generic solutions in the form of extensions to methods, languages, middleware and tools." [3] Component based architectures are used in the applications deployed on the middleware and these applications are reconfigured to meet the contextual requirements and provide the best utility for the users.

Aspect-oriented software development (AOSD) is an extension of object-oriented software development, and aims to maintain separation of concerns throughout the development process. All the code relating to a concern is kept in one module, the aspect, and new language constructs are used to control where the code is applied in the system. AOSD also provides powerful change mechanisms that allow additions or removals of aspects at run-time.

This emerging technology shows a promising potential for adaptive systems like the ones developed by the FAMOUS and MADAM projects. This paper evaluates the usefulness of AOSD in the context of adaptive mobile systems in general and the relevance to the FAMOUS and MADAM projects in particular.

The problem definition is summarized in Section 1.1 and the project context is presented in Section 1.2. Since Aspect-oriented software development play such a vital role in this paper we present a brief introduction to the basic ideas behind aspect-orientation in Section 1.3. For readers who are unfamiliar to AOSD concepts and terminology Section 1.3 is important to understand the later sections.

1.1 Problem Definition

The project description as formulated by SINTEF ICT prior to acceptance:

"FAMOUS - Aspect-oriented based service adaptation
When people are moving around using handheld networked devices, the operating environment for the provided services varies. User activities and location change, influencing user needs. Computer and network capabilities change, influencing service quality properties. Under such circumstances, services (and applications) should be adapted in order to overcome any mismatch between user needs and service properties, or between application needs and execution context.

The FAMOUS project at SINTEF ICT is developing a framework for building adaptive mobile ubiquitous services. In order to achieve adaptability, applications are constructed as component frameworks. We use properties in order to discriminate between alternative application variants. They are associated with components, and describe the properties of the services offered by a component, or the properties of the services needed by a component or the execution context needed by a component. The FAMOUS approach supports application adaptation and reconfiguration at the component level. We now intend to investigate techniques for adapting at a finer level. Aspect-Oriented Programming (AOP) techniques have been proposed as an approach to fine-grained adaptivity.

This assignment consists of integrating AOP in FAMOUS approach. A service prototype should be developed in order to assess the work."

We interpret the problem description to be the following: This project will explore the domain of aspect-oriented software design and apply the technology and design techniques to create a new version of the FAMOUS prototype using an aspect-oriented approach. We will investigate the feasibility of using AOSD and aspect-weaving as a mechanism for adaptation. This will be done by creating a prototype to manage aspect adaptation. The development process and the results should be evaluated to see whether it carries any distinct advantages over the current approach in the development of adaptive systems.

1.2 Project Context

This is the master thesis of John Christian Hveding and the conclusion of five years of computer science studies with a specialization in software engineering. The thesis is written at the department of Computer and Information Science at the Norwegian University of Science and Technology. The thesis is written for SINTEF ICT and the thesis is relevant to both the FAMOUS and MADAM projects that SINTEF ICT is participating in. The thesis is written in Trondheim, Norway.

The FAMOUS (Framework for Adaptive Mobile and Ubiquitous Services) is a strategic research program at SINTEF ICT funded by the Research Council of Norway. The project period is 2003-2006.

The Mobility and ADAptation enabling Middleware (MADAM) project is a 30 month European collaborative research project between Simula Research Laboratory AS, Norway; Birdstep Technology ASA, Norway; Condat AG, Germany; Hewlett Packard Italiana, Italy; Integrasys S.A., Spain; SINTEF ICT, Norway; Technische Universtät Berlin, Germany; University of Cyprus, Cyprus. The research is funded by the EC.

This thesis is a first step at exploring aspect-oriented software development in the context of adaptive systems. The thesis is not directly a part of the FAMOUS or MADAM projects, but its results could be relevant to both projects.

1.3 Introduction to Aspect-Oriented Software Development

To make it easier for the reader to understand what is going on in the later sections and since aspect-oriented software development is still a relatively unfamiliar development method to most, a short introduction to the subject is provided here.

Throughout the history of software engineering programming paradigms has evolved from being a direct mapping of the machine architecture (machine code and early development languages) to having a more problem-centric view. In this evolution object-oriented design is the current state-of-the-art. Object-oriented design tries to decompose the problem into units of primary functionality, while all other concerns are addressed in the code wherever appropriate. Aspect-oriented software development (AOSD) is a software development methodology that tries to address the non-primary concerns in a better way than object-oriented development practices.

Object-oriented design techniques try to compose the program into units with well-defined responsibilities. This often leads to the code of some concerns being spread throughout the system. Code tangling, having the code relating to different concerns tangled together, and code scattering, having the code to a concern scattered throughout several classes, are both issues familiar to most developers who have used object-oriented development methods. These are two of the things AOSD tries to counter.

AOSD address concerns by giving the developer a new code unit called the aspect. **Aspects** are a code unit that contains code pertaining to a specific concern. A **pointcut** or a **pointcut designator** is used to identify where the aspect applies. The pointcut is quantifiable and can refer to several different points in the system in a single statement. A point in the code where a pointcut can be applied is referred to as a **joinpoint.** Examples of joinpoints are method execution, method calls, reading a variable, assigning a variable or catching an exception. This quantification is one of the main advantages of aspect-orientation, and it enables AOSD to be particularly adept at handling **crosscutting concerns**, that is, concerns that apply to multiple units or are relevant to large parts of a system.

The code that is applied to a joinpoint is referred to as **advice**. **Introductions** or **intertype declarations** are a static addition to a class or interface. It does not directly change the behavior of the affected class or interface, but can add fields or methods to the advised class. Introductions allow fields and methods that are not part of the primary functionality of the object to be put into aspects that address these concerns.

An example of a concern that aspect-orientation is particularly adept at handling is logging. Consider an application that after deployment need to add a logging capability to all objects that access a particular resource. In traditional development the logging code would have to be added to every object that access this object, along with the code for opening the log-file and exception handling for writing to a file etc. In aspect-oriented design this logging code would be kept in one place only, separate from the base code, in an aspect, and a pointcut would refer to all the points where the logging would need to take place.

The process of aspect-oriented programming can roughly be separated into three steps:

- Identifying concerns (Design phase).
- Implementing the concerns as independent modules.
- Weaving the modules together.

The first step of identifying concerns through decomposition of the requirement specification is similar to any other development method, but with a focus of keeping concerns separate. Crosscutting concerns can be found both in the functional and non-functional requirements. There are several techniques for this. Jacobson presents a use case based method for aspect-oriented software design in [35]. There are also methods for including aspects in the requirements gathering, discussed in chapters 17 of [10]. Several ongoing projects in the AOSD community that underline the importance of taking aspects into account in the entire design process are described in chapters 17, 18, 21 and 23 of [10].

The core concerns are implemented in an object-oriented or procedural manner as in other development methods. The crosscutting concerns are specified in aspects which are maintained as independent code modules. The aspects also contain specification of where in the base code the advice code should be applied.

Finally the base code is integrated with the advice code through a process called **weaving**. Weaving is performed automatically by the AOP frameworks and can take place at compile time, load-time or run-time, the two latter also giving the developer a powerful dynamic tool.

Another AOSD approach follows a strict rule of never containing an element in object A whose sole purpose it is to make object B be able to use object A. This means that any mechanism that object A should contain to support object B is contained within object B and added to object A as introductions (intertype declarations) or weaved in as advice code. Only by following this strict rule can one eliminate the problems of code tangling and code scattering.

1.3.1 Features and Consequences of AOSD

AOSD creates a cleaner separation of concerns in the code. Separating the secondary concerns from the primary functionality results in objects with more well-defined tasks; modules contain code pertaining to their primary task and less code for other tasks. This leads to better maintainability, potentially more code reuse and less code duplication.

A cleaner separation of concerns makes the code relate more to the design descriptions of the systems. Prior to the advent of AOSD, developers had no way of transferring concerns separated in the design phase into the implementation phase. Separated concerns in the design phase could not prevent code tangling and code scattering. AOSD is a way for developers to keep separation of concerns also in the implementation phase of the project.

Developers are allowed to delay implementing certain concerns of the system that can be added later as aspects. The developer can focus on core concerns and new requirements can more easily be addressed as needed. This also allows requirements that pop up after deployment to be added with less effort. There is no need to change the existing code, just adapt the existing system to support the new features with an aspect.

The complexity of AOSD can cause an adoption overhead. As with any new technology there are costs in adopting a new tool or method in software engineering. In addition to this the code structure can also be considered more complex, the program flow of an aspect-oriented system is much harder to follow from reading the code as an aspect might completely bypass an entire method without leaving any hints about this where the method is defined or called. This is a breach of the principle of encapsulation which most OO-proponents consider sacred. The effects of this can be countered by using an integrated development environment (IDE) that supports AOSD.

Using don’t-care operators or wildcards can lead to undesired results if new joinpoints are added. A method in Java is identified by its name and arguments in combination. Overloaded methods, methods with the same name but different arguments, can cause unintended joinpoints to be matched by a pointcut designator, if new methods are introduced to the class receiving advice. Code that can become invalidated by non-local changes could pose a problem.

Several aspects might affect the same joinpoint and change the behavior of the system in this point. The effects of this might in some cases be unpredictable. For static systems this is less of a problem, any unintended consequences of the weaving would be discovered early. In dynamic systems however, this might cause a problem, as it is harder to discover because two conflicting aspects would have to be weaved in at the same time to discover the problem.

Some AOP frameworks contain mechanisms for run-time reconfiguration. This is a powerful and easy to use method for achieving dynamic reconfiguration of systems. Systems that offer dynamic weaving suffer a slight performance penalty in order to allow aspects to be weaved in or out of the base code.

1.3.2 Some AOP/AOSD History

The term “Aspect-oriented programming” was first coined by Kiczales in 1997, working at Xerox PARC, in an article with the same name [12]. Prior to this there were many works which also strove to separate concerns. Program slicing, meta-object protocols, compositional filters and adaptive programming all share some of the ideas behind aspect-orientation. The first works on aspect-oriented programming at Xerox PARC begat AspectJ. AspectJ became the first AOP framework.

Several other AOP frameworks popped up in its wake, each with different goals and focuses. As the field became more mature AOP was extended with a set of design methodologies, and the term Aspect-Oriented Software Development (AOSD) was used on the field to include also the design and modeling techniques unique to the aspect-oriented approach. The subject earned itself a community and an annual conference [16].

AOP and AOSD are buzzwords and have been hyped, possibly beyond its potential. As with many other new technologies in the field of software development, it is first hailed as the savior of the industry later to be criticized as harmful and finally be accepted as a tool to use on a particular class of problems.

More information on the basic concepts behind AOP is to be found in [11] and [12]. A Glossary containing some definitions and concepts is provided in Appendix B. A good reference on the state-of-the-art of the AOSD community is the book Aspect-Oriented Software Development [10].

2 Problem elaboration

Section 2 describes the project's goals and process. We begin with presenting a more detailed look at what the FAMOUS project is trying to achieve and how they go about doing it in Section 2.1. Five research questions that will be answered in the course of this project have been formulated in Section 2.2. The different research methods that will be used is discussed in Section 2.3 and finally the project is broken down into subtasks in Section 2.4.

2.1 The FAMOUS Approach

The FAMOUS approach to adaptivity goes through generic middleware and the technologies behind it are among other things component frameworks, variability engineering and architectural reflection. The description of the FAMOUS approach is mainly collected from [13] and [15].

The FAMOUS approach to creating more adaptive systems is to create an adaptive architecture. "To achieve true adaptability, the architecture itself will have to be adaptive." [4]. The FAMOUS project aims to create a generic middleware platform for adaptive mobile applications. The tasks of the FAMOUS middleware are:

- To collect, maintain, abstract and reason about context.
- To maintain a run-time model of the application.
- To evaluate all the possible configurations based on the context and decide what configuration is most suited to the current context.
- To reconfigure the application to best fit the current context.

Component Frameworks are "a dedicated and focused architecture, usually around a few key mechanisms, and a fixed set of policies for mechanisms at the component level" as defined by Szyperski in [14]. Additionally, components that can be added to the component framework to alter its behavior and the rules governing such reconfigurations are also considered to be part of a component framework in the context of FAMOUS. The applications are implemented and deployed on the FAMOUS middleware as component frameworks.

Variability engineering is the art of creating systems that has to meet requirements that vary in the life of the system. Dynamic requirements demand an adaptable application that can have various compositions to meet the variable requirements. Variability engineering is used to create application as component frameworks with various possibilities of adaptation.

When a system reasons about its own internal state, it is called reflection. In FAMOUS the middleware maintains a model of the application at run-time. This model is modified and evaluated to find the composition of the component frameworks of the application that is best suited to the current context. The running version of the application is then modified to the configuration that the middleware decides will generate the highest level of utility for the user, based on the evaluation of the meta-model.

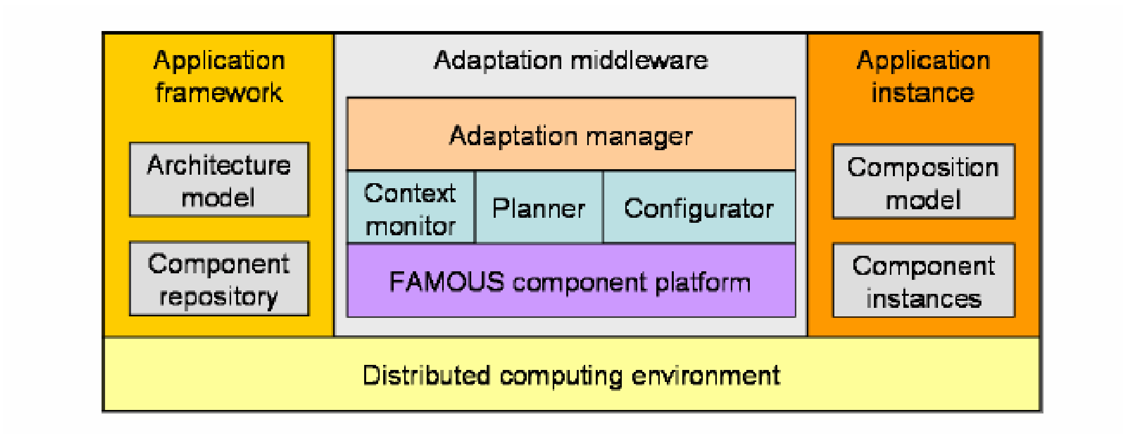

Figure 2.1 – The FAMOUS Architecture

Figure 2.1 depicts the architecture of the FAMOUS middleware platform. The Context monitor monitors the context, maintains context data and notifies the other components of context changes. The Planner uses the Architecture model and the Component repository to build different configurations of the application and identify the most suitable composition. If the Planner finds a configuration that is sufficiently better suited to the current context than the one that is currently running, the model is passed to the Configurator. The Configurator then reconfigures the Composition model of the application.

The task the Context monitor is to keep track of is of two different types of context: User context and network context. The former covers context elements that are related to the user and the environment of the user. Examples of user context are activity, location and role of the user or lighting conditions of the environment of the user. Network context covers elements of the device and underlying infrastructure. Examples of network context are available memory of a device or the bandwidth of a particular connection.

The model kept by FAMOUS middleware is a composition of the various components of the application. These components are connected through ports that define a service offered by the component or a service consumed by the component to produce a service. The middleware abstract the basic services provided by the device and the services consumed by the user. There is support for composite components that can be broken down to several sub-components. The model also contains the notion of nodes on which components can run. This allows parts of the application to be run on nodes other than the mobile device itself.

The actual variation in the model is achieved by having several different sets of components that offer or use the same services but with different properties. Composing a configuration consist of selecting a set of components that fit together to produce the services required by the user and consume the services provided by the underlying system. Moving components to other nodes and adding new components to take advantage of newly discovered resources are other examples of modification of the application which FAMOUS allows.

The components are connected through ports. To the ports of the components there are associated values which represent the “amount of service” they require or provide. These properties can be integers, with or without an associated range; Enumerations, with a list of allowed values; Booleans, true or false or Strings. These ports consume or provide a set of services. Figure 2.2 illustrates how the components are modeled. The components implement a role and offer services to each other or to the user, and consume each others' services or the services provided by the underlying system. Figure 2.2 is taken from [13].

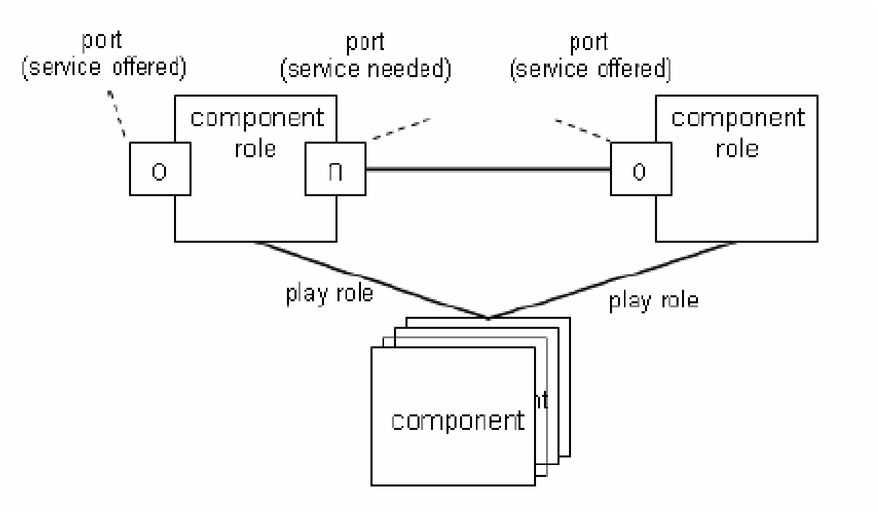

Figure 2.2 – Component Roles and Ports

To the property of each port there is attached an evaluation function. This function evaluates the level of service offered based on how well the component's needs are met. To evaluate the application there is a utility function which evaluates how the application as a whole meets the user's needs in the current context. This utility function is the basis of all reconfigurations. The system chooses the configuration that gives the highest utility in the current context.

The FAMOUS middleware respond to context changes by evaluating the need for adapting the application to the new context. So every time a context change occurs, the FAMOUS middleware evaluates the utility of all configurations. If there is another configuration that returns a significant improvement over the current, the architecture of the application is changed. This reconfiguration is a relatively coarse-grained change, which is one of the reasons why this thesis will be looking at aspect-orientation as mechanism of adaptation.

2.2 Research Questions

Based on the challenges discussed in the previous section and the potential capabilities inherent in AOSD we have formulated a set of research questions to focus this thesis around. This project is an explorative project to explore the domain of aspect-oriented design methods in the context of adaptive systems.

The main goal of this thesis is to answer the question:

Is aspect-oriented software development a suitable development technique for development of adaptive systems?

This question will be answered indirectly through these sub-questions:

1) Are aspect-oriented programming mechanisms for run-time adaptation better suited than component framework reconfiguration?

2) Are aspect-oriented programming mechanisms for run-time adaptation better suited at adapting fine-grained adaptations than component frameworks?
3) Are aspect-oriented programming mechanisms for run-time adaptation better suited at adapting cross-cutting concerns than component frameworks? Are crosscutting concerns commonly reconfigured in adaptive systems?
4) Does aspect-oriented software development lead to less and simpler meta-information about the (reconfiguration of) the application?
5) Is the current FAMOUS utility evaluation technique for component reconfiguration applicable for aspect adaptation?

2.3 Research Method

One usually distinguishes three ways of doing software engineering research (from [38]): The engineering experimentation method is where one creates a running system and tests it based on a hypothesis. Based on these tests the system is improved. The empirical method is where one uses statistical methods to validate or falsify a hypothesis. The analytical method is where one has a formal model of the system one creates, and compares the result of the model with that of the empirical observations. Combinations of these three methods are also possible. The engineering experimentation method is the basis of this thesis.

From the twelve different approaches to software engineering described in [39], we are using two: Assertion and Literature search.

An assertion is an observational validation method where an experiment is developed and executed to evaluate technology and compare it to other technologies. It is usually executed in an ad-hoc manner. This thesis uses assertion as our project evaluates whether the new technology AOSD is applicable to adaptive system and compares it with alternative approaches.

In order to become more acquainted with such a huge and recent field of study as aspect-oriented software development is, we started with a literature search. We began the literature search with three books on AOP/AOSD. In addition to the three books, we have read a number of articles relating to the AOP/AOSD field. We also had to become acquainted with the FAMOUS/MADAM documentation, which covers six deliverables, two published articles and some internal SINTEF documents we were given access to. The result of the literature search is presented in Section 3.

As we became familiar with the domain of adaptive systems and the techniques of aspect-orientation we created two prototypes of an adaptive system based on aspect-oriented mechanisms. After designing and implementing these prototypes we discuss our experiences of using aspect-oriented techniques in creating an adaptive system. We finally discuss how our approach compares to the work done by the FAMOUS team.

2.4 The Subtasks of the Project

To answer the research questions formulated in Section 2.2 we created an aspect-oriented adaptation-prototype based on the scenarios described in the FAMOUS documentation. But in order to do so we first had to study the aspect-oriented programming techniques and identify the best tools for getting the job done. During the process to see if an aspect-oriented approach to adaptation is possible, we gathered experiences that are discussed later in this paper.

2.4.1 Learning AOSD

A prerequisite to creating a aspect-oriented system is to become acquainted with AOSD. The aspect-oriented programming paradigm, aspect oriented design practices and available aspect-oriented programming tools are all topics which need to be mastered to complete this project. To get a grip on the topic of aspect-oriented software development we started by looking at articles published at the annual AOSD conference to see if we could find articles relevant to dynamic adaptation and adaptive systems in general [16]. We also identified several books relevant to our project and studied these.

To fully get a grip on the aspect-oriented programming paradigm we created a small application that performs some basic weaving operations. The application consists of three classes, two for the base application and one aspect. The task of the application is to generate two random numbers between 1 and 6 and present these to the user every time the “roll dice”-button is pressed by the user. The numbers are presented as two digits in a text field. The role of the aspect is to change the presentation of the numbers from digits to dice-icons. This is achieved by weaving in the advice code “instead of” the base code that output the numbers.

2.4.2 Choosing the Relevant Implementation Tools

The current FAMOUS implementation is compatible with both J2SE and J2ME (CDC, Personal Profile). To limit the complexity this project will be implemented in J2SE without explicitly supporting J2ME. Although the FAMOUS project’s objective is to create mobile middleware, the task of this project is to evaluate a new development paradigm and on what platform that is achieved is without consequence. As the current AOSD support on J2ME is scarce, it saves us time and effort to implement the prototype in J2SE.

The AOP framework we choose to use should fulfill certain requirements. There are many different frameworks and they all focus on different issues. We need a framework that provides dynamic aspect weaving but also, due to the limited timeframe of this project, a framework that we can learn to use in a relatively short time.

Requirements of the AOP Framework:

- As all work on FAMOUS is currently implemented in Java it is natural that the AOP Framework also be Java based.
- The AOP framework should support dynamic aspect weaving at run-time or load-time.
- The AOP framework should be relatively mature and have a high probability of being further developed and maintained in the future.
- The AOP framework should be easy to familiarize with and use.
- The AOP framework should be well documented and this documentation should be up to date.
- The AOP framework should have a developer forum and should preferably be used by a wide developer base.

A number of different approaches have been suggested for modeling aspect-oriented systems. However, a unified method does not currently exist. Different proposals of methods for modeling AOP-systems were looked at. Of particular interest would be finding a method of modeling crosscutting concerns and finding a method of modeling run-time weaving of aspects.

2.4.3 Design

Based on the possibilities inherent in AOSD we described a strategy for implementing FAMOUS in an aspect-oriented approach. Based on the scenarios and the design done by the FAMOUS team, we created use cases to model the wanted behavior of our application. To these use cases we added the wanted adaptations.

2.4.4 Prototyping

After becoming familiar with aspect-oriented programming and design practices we created two prototypes which use an aspect-orientated adaptation mechanism. These prototypes mimic the role of the FAMOUS middleware, but do not match its complexity level. The main goal of the prototype is to prove that an aspect-oriented adaptation is possible.

2.4.5 Collecting Experiences

Throughout the whole development process we collected experiences on the advantages and potential disadvantages of using aspect-orientation in the development of adaptive systems. We evaluated the prototype and the development process and compared this with the existing implementation and development methods to determine whether AOSD is beneficial in the development of adaptive system.

2.4.6 Additional Work

As the aspect-oriented approach keeps meta-information about what an aspect can do implicitly, and the FAMOUS middleware keeps an explicit meta-model of the different configurations, if time permits, an extension to the project would be to investigate whether these two meta-models can be integrated.

3 State-of-the-art

New versions of AOSD tools are released to the market continuously, and a lot of research is taking place relevant to the AOSD community. Some of the information in this paper is already becoming less accurate due to this rapid progress. AOP frameworks have been developed by many different institutions with diverging research goals. This has led to many small implementations. However, after the initial burst of diverse small frameworks, the market is now converging on a few standards.

AOP alliance [17] is a project to standardize AOP. Many prominent people behind some of the most important AOP frameworks support this initiative. The goal is to create a standard API for aspect-oriented programming. The project is still in a very early stage, and consists of a set of interfaces and a discussion forum. But it may lead to a JSR (a Java Service Request) and eventually a minimized API that all the AOP frameworks which are part of the alliance would support.

Since aspect-orientation is such a buzzword it can be quite a challenge to find validated information on it. After a thorough analysis of the many books on the topic available on a large unnamed online bookstore we chose three books.

As our initial analysis lead us to believe that AspectJ would be the most suited AOP framework we got the book “AspectJ in Action” [9]. It provides a basic introduction to the field of AOP, an in-depth look at aspect-oriented programming in general and AspectJ in particular. As we later chose use JBoss AOP we had little use of this book other than as a primer for Aspect-oriented programming.

The book “Aspect-oriented software development” [10] had in its contributor list many of the most prominent experts in the field of AOSD. The book is a collection of articles by many different experts on the field of AOSD. The book grew out of the Communications of the ACM October 2001 aspect-orientation special, and has later been updated with input from researchers from many fields within AOSD. It is the most comprehensive book on AOSD to date.

The book “Aspect-oriented software design with Use-cases” [35] by Jacobsen and Ng sees aspect-oriented programming techniques as the missing link of use case based development. Use-case development already provides a separation of concerns in the design phase of a project, but it has until the advent of AOP been impossible to keep the concerns separate in the implementation phase. The book contains use case designing guides and how this relates to aspect-oriented design. Additionally it suggests a method for modeling crosscutting concerns with use cases.

In addition to the three books, we have read a number of articles relating to the AOP/AOSD field. These articles were found by looking up references from the books, searching Google for AOSD/AOP and following links from the different AOP frameworks community pages. Some articles were suggested to us by our advisor.

We try to describe the current state of the AOSD community starting with a discussion on AOP frameworks in Section 3.1. Section 3.2 deals with different methods of modeling aspect-orientation. In Section 3.3 we present a set of views on AOSD design principles that might be relevant to this project.

3.1 AOP Frameworks

In order to take a closer look at the different AOP frameworks it is important to understand what is going on under the hood of an AOP framework. To describe the mechanisms that are used to implement AOP we make use of the common terminology used in the AOSD community. It is therefore recommended that readers unfamiliar with AOP read Section 1.3 to acquaint themselves with the basic AOP terms and concepts. The aspect-oriented software development community [16] lists 30 different AOP frameworks as “supported”, of which about half are for Java. As only Java based systems are relevant to this project. The most promising candidates are discussed below, after a brief summary of the basic technology used to implement AOP.

In AOSD, aspects are weaved into the base code to create a complete system. This weaving-process can be static; performed at compile-time, or dynamic; performed at load- or run-time. While static systems still benefit from the separation of concerns that AOSD brings, it is the dynamic systems that contribute a powerful new tool to use in adaptive systems. As a consequence our focus is on dynamic systems.

Static weaving is the integration of aspects with the core classes at compile-time. The weaving can take place before compilation, in a pre-compiler. This is the method chosen by the AspectJ group (see Section 3.1.1). As they have extended the Java language with new language constructs, the code has to be pre-compiled before it is fed into a Java compiler. Alternatively the weaving can take place after a standard Java compiler has converted the program to byte-code. The byte-code is then manipulated with a byte-code manipulator to weave in the aspects.

There are three basic solutions that achieve dynamic weaving in use in current AOP systems:

- Load-time weaving
- Just-in-time (JIT) compiler weaving
- Using the Java Reflection API

Frameworks using load-time weaving either replace the root class loader in Java, or subclass from the Java abstract ClassLoader class. The latter is a standard Java mechanism, which means that it is supported within the “normal” use of Java. The former requires changes to the Java virtual machine or means running the Java virtual machine with special arguments or in debug-mode. Replacing the root class loader is a breach of the Java security mechanism, but is necessary if the AOP developer is to be able to advise system classes.

Frameworks using JIT compiler weaving perform no byte-code alterations. The weaving is delayed until the byte-code is to be compiled to native code in the Java virtual machine. This requires extensive changes of the Java virtual machine. Frameworks using JIT compiler weaving add hooks directly into the native code when the transformation from byte code takes place. These hooks make a call back to aspect manager that determine whether advice code should be executed at this joinpoint.

A third approach to dynamic AOP frameworks is to use the Java Reflection API. The Java Reflection API lets developers reason about the internal structure of the program and can create dynamic proxies that wrap around certain methods or classes to intercept and insert advice code certain places in the code. The Java Reflection approach is a standard Java mechanism and therefore requires no change to the Java virtual machine and can use any standard Java compiler.

Load-time weavers and JIT compiler weavers also need to mark the joinpoints in the code to be able to apply aspects there. These marks are usually referred to as hooks. The hooks make a call to the aspect manager that checks whether a pointcut refers to the hook in question, and executes the advice if that is the case. Three hook-policies can be used to do this:

- Total hook weaving adds a hook to all joinpoints.
- Actual hook weaving adds hooks to joinpoints which are defined to be candidates for weaving.
- Collected weavers weave the applied code directly into the base code.

The AOP frameworks have different joinpoint models (the joinpoints supported by framework). Additionally the advice code that is applied to a joinpoint can be applied before, after, around or instead of the joinpoint. The before and after modifiers are self-explanatory, while around lets the developer add code before the joinpoint, then decide whether and when the joinpoint should be executed and then add code after the joinpoint is executed. The instead modifier is a special case of the around modifier where the original joinpoint is not executed. The frameworks support different versions and combinations of these modifiers.

3.1.1 AspectJ

One of the most widely known aspect-oriented languages is AspectJ [18]. It was developed at PARC and was later transferred to the Eclipse Foundation which now maintains and develops AspectJ. The developers chose to add new keywords to Java to implement AspectJ with the consequence that AspectJ code needs to be pre-compiled, or compiled with a compiler that supports these new keywords. The AspectJ compiler is called ajc and is a pre-compiler that feeds it results directly into the standard Java compiler (javac).

AspectJ has until now primarily been focused on compile-time weaving, but later versions also support load-time weaving. The load-time features are as of now only available in unstable releases. Full support for load-time weaving should become available with the stable release of AspectJ 5. Despite currently lacking support for dynamic weaving of aspects, the future looks bright as the Aspectwerkz team (see Section 3.1.4) has joined forces with the AspectJ team, and they are now collaborating to produce AspectJ 5.

AspectJ has the larges joinpoint model of all the Java based AOP frameworks and has a powerful syntax for defining pointcuts. AspectJ syntax is very similar to Java syntax and is a natural extension of Java that most Java developers would feel at home with. All pointcut declarations are made with this Java/AspectJ syntax and there is no need to define pointcuts in separate XML-files.

Even though AspectJ will probably dominate the aspect-oriented language scene in the near future, its current support for dynamic weaving is insufficient to our needs. AspectJ can easily be integrated with the Eclipse foundation’s Java IDE. AspectJ is an open source project released under the Common Public License (CPL).

3.1.2 PROSE

The ***PRO**grammable exten**S**ions of s**E**rvices (*PROSE) [19] was developed at the Swiss Federal Institute of Technology Zurich with a goal of addressing dynamic AOP. Unlike most AOP frameworks prior to PROSE, which mostly used load-time modification of the program to achieve dynamism or only provided static weaving, PROSE focuses on run-time weaving.

The first version of PROSE was based on the Java Virtual Machine Debugger Interface (JVMDI) in Java 1.2 SDK [21]. The JVMDI allows developers to inspect the state of the JVM and register execution request at certain points. The later PROSE version uses JIT compiler weaving and is based on the IBM Jikes Research Virtual Machine [22]. This makes the later versions of PROSE less portable as they are tightly coupled with a specific modified JVM. PROSE also comes distributed with an updated version based on the JVMDI to use on other JVMs. PROSE can be configured to use either actual or total hook weaving. The developer specifies to the system which classes are candidates for weaving (or which are not) and hooks are weaved into all joinpoints in all specified classes (or all classes not specified). The aspect code is kept separate from the base code.

An interesting aspect of PROSE is that it aims to perform the weaving operation atomically. This means that the relevant hooks are blocked until the weaving is completed. It is also possible to group several weaving or unweaving operations together and prepare these but delay committing them. This would make it possible to control the synchronization of the weaving in distributed systems.

The joinpoint model of PROSE is rather rudimentary. Methods, exception throwing, variable reading and writing are supported as joinpoints. PROSE is a pure Java AOP Framework. Aspects in PROSE are normal classes which extend the PROSE framework classes. Systems developed in PROSE are therefore compiled using a normal compiler. An example of a system that uses PROSE's mechanisms for run-time weaving to achieve dynamic adaptation in middleware is presented in [23]. PROSE is an open source project, released under the Mozilla Public License.

The documentation provided with PROSE [20] is relevant for version 1.0.2 and seem to be little updated after that, while the current version of PROSE is 1.2.1. There is a discrepancy between the documentation and the executables provided.

3.1.3 JBoss AOP

JBoss is an open source application server. The company behind it is also responsible for JBoss AOP [24]. JBoss AOP was developed to provide a useful extension to JBoss the application server and particularly to be able to implement changes to the server at run-time. It is also possible to run JBoss AOP in stand-alone mode, without the application server. JBoss AOP also includes an aspect library which supplies reusable solutions to a set of generic crosscutting concerns.

JBoss AOP uses load-time weaving through a custom class loader. JBoss can be configured to use either actual hook weaving or total hook weaving. The custom class loader (called unified class loader), loads all classes into a flat namespace, where all classes can reach each other. JBoss uses XML to bind pointcuts to "interceptor"-classes.

The joinpoint model of JBoss is feature rich and has a powerful pointcut syntax. JBoss is a pure Java AOP framework and can be compiled by any Java compiler. Aspects in JBoss (sometimes referred to as interceptors), extend the JBoss AOP framework classes. JBoss is easily integrated with the Eclipse platform. JBoss is released under the GNU Lesser General Public License (LGPL). More details about the JBoss architecture can be found in [27].

3.1.4 Aspectwerkz

Aspectwerkz [28], created by Jonas Boner and Alexandre Vasseur, is a lightweight AOP framework. "Aspectwerkz is a dynamic, lightweight and high-performance AOP framework for Java", from [28]. Aspectwerkz is an open source project, being maintained at codehaus, and is developed in collaboration between many developers.

Aspectwerkz provides dynamic weaving through load-time actual hook weaving with the help of the Hotswap classloader. Aspectwerkz is easily integrated with the Eclipse platform. Like JBoss Aspectwerkz also use XML to define pointcuts. Aspectwerkz features a rich joinpoint model. Aspectwerkz is released under the LGPL.

3.1.5 JAC

JAC (Java Aspect Components) [29] is based on Renaud Pawlak's PhD thesis of 2002 [30]. It was developed by AOPSYS in collaboration with the Laboratoire d'Informatique de Paris 6 (LIP6), Laboratoire CEDRIC and Laboratoire Lifl.

JAC uses load-time weaving with a custom class loader to achieve dynamism. The byte code is altered with the byte-code engineering library BCEL. JAC uses an actual joinpoint policy.

The joinpoint model of JAC is very narrow, only method executions and method calls are supported. JAC is a pure Java AOP framework and can be compiled by any Java compiler. JAC is released under the LGPL. JAC also consists of a generic aspect library and a console for on-the-fly weaving of aspects in running applications. JAC is also described in Chapter 16 of [10].

3.1.7 A Comparison of Dynamic AOP Frameworks

An article by Chitchyan and Sommerville [31] discusses how four AOP frameworks supporting dynamic weaving compare. Some of the descriptions of the AOP frameworks above are collected from this article. It discusses the relationship between the different hook policies, and states that the more hooks the more evolvability and flexibility, but also more performance overhead and complexity. Total hook weaving carries the highest run-time overhead in normal operation due to the processing of unused hooks. It is the most evolvable policy as aspects can be weaved in at any joinpoint without augmenting the underlying base code. The process of weaving is however more efficient in total hook weaving systems as no adding of hooks to the advised code is necessary. Actual hook weaving carries lower overhead in normal operation but suffers a significant response time penalty when adding advice to a base code unit without hooks, as this unit would have to be re-loaded. The collected weaving policy is not much used in dynamic AOP systems as the methods for removing advised code that is mixed with the base code is so complex.

It also provides an overview of the different techniques that can be used to achieve dynamic weaving and discuss their advantages and disadvantages. It further discusses how the four systems compare based on six criteria gathered from relevant works on dynamic reconfiguration. It is acknowledged by the authors that whether the results in the article are based on the different technological solution chosen by the respective framework or is simply a result of the implementation specifics or maturity level of each framework is unclear. We therefore choose to only report the performance discussion.

3.2 Modeling Aspect-Orientation

There is currently a plethora of different position papers regarding how to model aspect-oriented systems. Several articles which present different approaches have been studied; [32], [33] and [34]. There is no consensus between AOSD researchers on how AO should be modeled other than that it should be based in smaller or larger part on UML. In the book Aspect-oriented software design [10], Chapter 20 presents a method of modeling AO system within the constraints of UML, called implicit weaving, while Chapter 19 presents Theme/UML which is a way of extending UML to better model AO systems.

3.2.1 Implicit Weaving

The advantage of using implicit weaving is that one can use standard UML without modification for modeling. As there is a great many modeling and CASE tools available supporting UML this can be a significant advantage. Although the syntax of UML is used in this approach, the semantics is a little different, and some of the more obscure parts of UML become very central to the modeling. Additionally dynamic weaving is not explicitly supported. Implicit weaving is a inventive way of modeling separation of concerns and weaving within the confines of UML 2.0. We do however find it too limiting and not fit for our purposes.

3.2.2 Theme/UML

The Theme/UML model has its theoretical fundament in multi-dimensional separation of concerns. Different concerns are modelled and weaved together at a later stage. Although very good at decomposing the different aspects of an application, and displaying these, we found no support for modelling run-time weaving with this approach. As tool-support is limited and the complexity of the approach is rather high we decided not to use Theme/UML.

3.2.3 Using Use Cases to Model adaptations

Jacobson, in his book [35], sees aspect-oriented programming techniques as the missing link of use case based development. Use-case development already provides a separation of concerns in an early phase of a project, but it has until the advent of AOP been impossible to keep the concerns separate in the implementation phase. The book contains use case designing guides and how this relates to aspect-oriented design.

The use case extend relationship allows developers to add new functionality to existing use cases. The extending use case is described separately from the use case it extends. This being a modeling parallel to what one does in AOP. An extension point defines a point in the use case that can be referred to by other use cases. When the use case is changed, the extension point is also updated to refer to the same point. This way the use cases that refer to the extension point does not have to be updated every time a change is made to a use case. Jacobson contends that use case extensions map very well to aspects in the implementation phase.

3.2.4 Our Modeling

We have been unable to find any modeling technique that enable us to model run-time changes to an application, either through weaving or other mechanisms. We have decided to go with Jacobson’s use case approach to model adaptation. Although use cases are generally rather informal we believe that it is sufficient to convey our ideas of what we want to do. We model adaptations as extensions of the use case we wish to adapt.

3.3 AOSD Design Principles

As we mentioned briefly in Section 1.3 there several ways a project can benefit from AOSD. There are those who bring aspects into the picture in the in the later stages of OO-development. This is typically projects that want the benefit of the new technology of aspects, but cannot or does not want to use the aspect-oriented design methodology in the entire development process.

The other approach to AOSD follows a strict rule of never containing an element in object A whose sole purpose it is to make object B be able to use object A. This means that any mechanism that object A should contain to support object B is contained within object B and added to object A as introductions (intertype declarations) or weaved in as advice code.

Both methods benefit from AOSD, but only by following the strict rule can one eliminate the problems of code scattering and code tangling. Analysing the design with an aspect-oriented perspective or using aspects to encapsulate certain aspects in the implementation phase has its benefit, but it is only when the full scale of AOSD is used when one can get the full benefit of it.

3.3.1 A Project using PROSE to implement adaptive systems

The group behind PROSE has been using PROSE to create an adaptive system. Their efforts are described in Chapter 31 of [10]. In this project they suggested the roles of aspect bases and aspect receivers. An aspect base is a node that contains an aspect repository. This repository contains a set of aspects which are applied to the aspect receivers.

They describe a scenario which takes place in manufacturing systems. An unidentified fault has caused a quality drop in a section of the plant. To solve this problem mobile agents (robots) that operate in this section are required to log all their activities. This functionality is not included in the mobile agents and will have to be added. An aspect base is associated with a base station of the wireless network in this section of the plant. This aspect base contains an aspect with the wanted functionality. The aspect is weaved into the mobile agents at run-time without taking them offline. The mobile agents here operate as aspect receivers. When the mobile agents leave the troubled section the aspect is no longer needed and is discarded.

The roles of aspect base and aspect receivers can be distributed among the entities of the system. “At one extreme, each node can contain an aspect base.” (from chapter 31 of [10]). They envision a system where each node that joins an adaptive system distributes aspect which advice other nodes on how to use its services.

4 Own Contribution

In this section we present our ideas and our independent work. We give an outline of our ideas on how to solve the problem in Section 4.1. In Section 4.2 we take two scenarios and create a design for an application which can be implemented. The tools that are to be used for our implementation are discussed in Section 4.3. The design and implementation of our prototypes are presented in Section 4.4. Finally we present a test-run of one of our prototypes in Section 4.5.

4.1 Designing the System (higher level description of solutions)

In this section we present a description of our solution in broad terms. We described in Section 3.3.1 a project that uses aspect-oriented methods for adaptive system. They use aspect repositories to choose aspects from when creating adaptive systems. Similarly, as described in Section 2.1, the FAMOUS project uses a component repository that the planner can choose components from when evaluating different configurations. We find it naturally to have an aspect repository where all aspects are located, and which the adaptations system can access to retrieve aspects to adapt the application.

4.1.1 Using Aspects to Implement Adaptation

We believe aspect weaving is ideally suited to perform adaptations. It is a mechanism that can be performed at run-time. It is designed to handle cross-cutting concerns which often appear in adaptations. It eases the construction of adaptive systems by providing a powerful change mechanism embedded in a framework.

One of the problems arising from using aspects as a change mechanism is that aspects are not independent of each other. Their effects are not always orthogonal and when two aspects affect the same object, the result might not always be what the developer intended. The solution to this problem is either to construct aspects that do not affect each other, or make the middleware aware of which aspects are incompatible, and consequently not weave these to aspects in simultaneously. The former might not always be possible or desirable, so a mechanism that prevents incompatible aspects from being weaved together is preferable. In either case it is important to be aware of the issue and not develop aspects for different concerns completely independently.

A similar problem arises in component reconfiguration; not all configurations are valid. The FAMOUS team proposes to deal with this problem by defining constraints that prevents the system from using invalid configurations. A similar approach would be possible while using aspect adaptation.

4.1.2 Evaluating Aspects

It is possible to change the current evaluation system to take aspects into consideration. We here suggest three different approaches.

Aspects are represented in the model as components. This means all aspects implement one of the component interfaces and ports and properties are defined for all aspects. All components that have aspects that affect them can be modeled as composite components and the affecting aspects are sub-components of this composite component.

This approach requires little or no change to the actual adaptation system. It does require that one define values of the services provided and needed by the aspects. This is not always easy since aspects often embody non-functional requirements. It leads to more fine-grained components which enables more configurations. It also leads to more complexity of the model. Numerous components and configurations will of course lead to a time-consuming evaluation if no improvements in evaluation algorithms are done.

Another approach that requires little or no change to the evaluation system is to represent a component with aspects as several different components. Each current component would then be represented in the model's component repository one time for each combination of aspects that affects it. Example: a component has two aspects which affects it. In the model's component repository this would be represented as one component without influence of any aspects, one representation for each of the two aspects and with one representation for when both aspects are weaved in.

The FAMOUS planner can then choose from a large list of components with similar properties. The actual adaptation need not be done by reconfiguration. If component A is to be replaced by component A', the same component with an aspect, the aspect is just weaved into that component.

While this approach could be implemented without major changes to the evaluation system, it would lead to an even larger number of components than the previous approach. This would put an even larger strain upon the performance of the evaluation system. It would not require putting any values of on the aspects, and achieve the same fine-grained adaptations as the previous approach. The complexity of the model would be larger than the previous approach.

The third approach we suggest is to create a new meta-model that is specific to aspects. This aspect evaluation could be used to implement smaller adaptations. An application can be modified in many ways without changing its architecture. This "lower level" adaptation could be combined with the component evaluation by performing it before a component evaluation and reconfiguration takes place. If the aspect-adaptation achieves sufficient utility – go with that, if not – move on to component reconfiguration.

Whereas the two first approaches leaves the evaluation system largely unchanged, the third approach is a break with the FAMOUS evaluation system. We suggest a using simpler model to evaluate aspects. There are two reasons for choosing a simpler model. As described in Section 2.1, the FAMOUS prototype contains one evaluation function for each configuration. As the theoretical number of configurations in aspect adaptation can get quite high, this is not a practical way of evaluating the model. The second reason is simply the amount of work needed to get working prototype with the same complexity level of the FAMOUS prototype is somewhat beyond the number of man-hours in this project.

Independent of what model is chosen to decide which adaptations take place, we suggest making use of AOSD's crosscutting abilities. Any changes to the model caused by an adaptation could be weaved into the model in parallel with the actual adaptation. This approach is particularly suitable if the meta-model and the evaluation algorithms are as complex as in the FAMOUS project. This would also allow aspects which were not considered at deploy time to be more easily integrated with the system. Using an approach were the effects of a change is weaved into the model at the same time the actual change is weaved in to the application will open the door to more adaptive systems.

4.2 Scenario Descriptions

Several motivational scenarios were defined as part of the FAMOUS project. Our scenarios are collected from The Janitor Scenario. This is the same scenario that the FAMOUS prototype, version 1.0, is based on. The full text of the scenario is available in "FAMOUS deliverable D1.1 - v1 Context Model" [2]. As part of the FAMOUS project a design based on these scenarios has been made and are also reproduced here. The use cases presented are from "FAMOUS deliverable D3.3 – v1 Prototype service" [7].

The foundation for the scenarios are this (from [2]):
"The employees of a company reports through P&P (aka "Pompel and Pilt") reporting system about excessive warmth in their office. A janitor working on site close to the requesting company is called in to control and adjust the temperature.

4.2.1 Scenario Description "Preparing the work"

The specifics of the first scenario:

"***Preparing the work***
• *In order to perform the inspection, the janitor requests information about the equipment to be inspected and an installation map. Some application for guiding the recording of the temperature measurements is also needed. Information and applications are available from the requesting company equipment server. In order to reduce communication costs, downloading will take place when the janitor gets access to the company WLAN (free of charge)."*

Based on this scenario the FAMOUS designers made a use case to specify what their system is supposed to do. Their use case is depicted in Figure 4.1 and the associated textual descriptions are listed in Table 4.1.

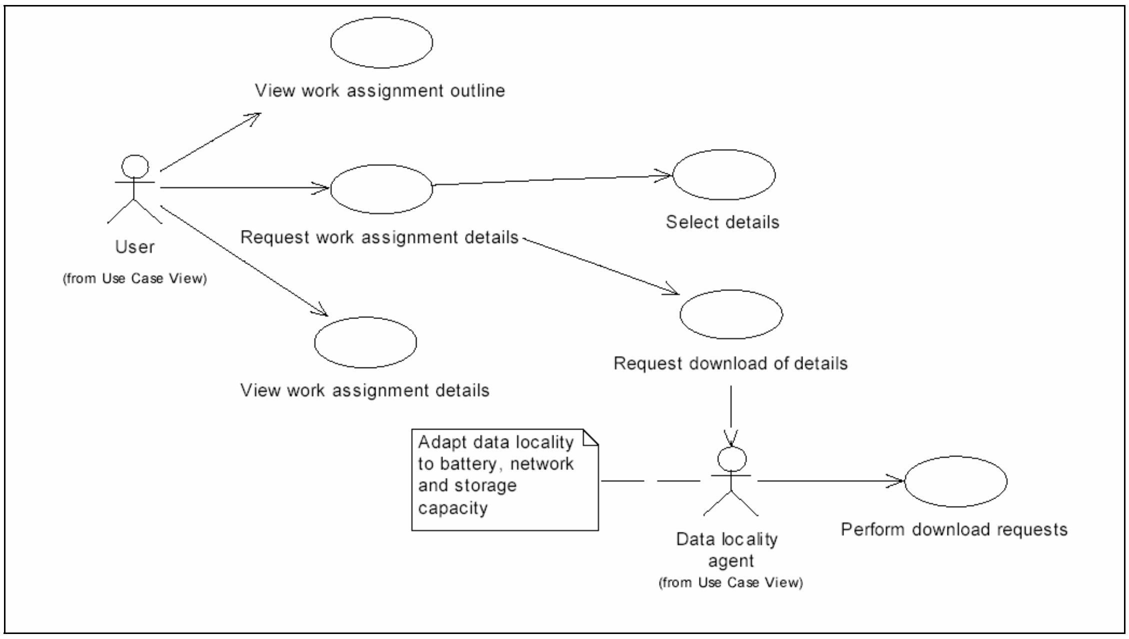
View work assignment outline
User
(from Use Case View)
Request work assignment details
Select details
View work assignment details
Request download of details
Adapt data locality to battery, network and storage capacity
Data locality agent
(from Use Case View)
Perform download requests

Figure 4.1 – Use Case "Preparing the work"

Table 4.1 – Use Case “Preparing the work” from FAMOUS

Use case	**Comment**	**Adaptation to context**
View work assignment outline	The work assignment outline is stored on the client.	
Request work assignment details	The janitor may already be familiar with some information elements. For example, a janitor familiar with the building may not need a map of the building.	
Select details	The importance of information should be given (see below).	
Request download details		
Perform download requests	The data locality agent administrates downloading: - Some information may already be available on the PDA. - Downloading is adapted to context.	Limited storage: the most important information is downloaded. Network conditions (bad coverage, expensive use): downloading is postponed
View work assignment details		

We reworked the use case to be based on the principles from [35]. We started with the use case made by the FAMOUS team and first stripped them to only model a typical case or best-case scenario (where no adaptation is necessary). We then added a set of “extension points” as defined by Jacobson in [35], where adaptation might occur. An extension point defines a point in the use case that can be referred to by other use cases. When the use case is changed, the extension point is also updated to refer to the same point. This way the use cases that refer to the extension point does not have to be updated every time a change is made to a use case. Jacobson further contends that functionality modelled as use case extensions naturally correspond to aspects in the implementation phase.

In our use case figure we have chosen to abstract away all the adaptations into an aspect library. The aspect library is represented as a stack of use cases as it contains several different adaptations. This is to make the figure more manageable. The adaptation use cases are represented in Tables 4.3 - 4.5. After following this procedure we ended up with the use case depicted in Figure 4.2.

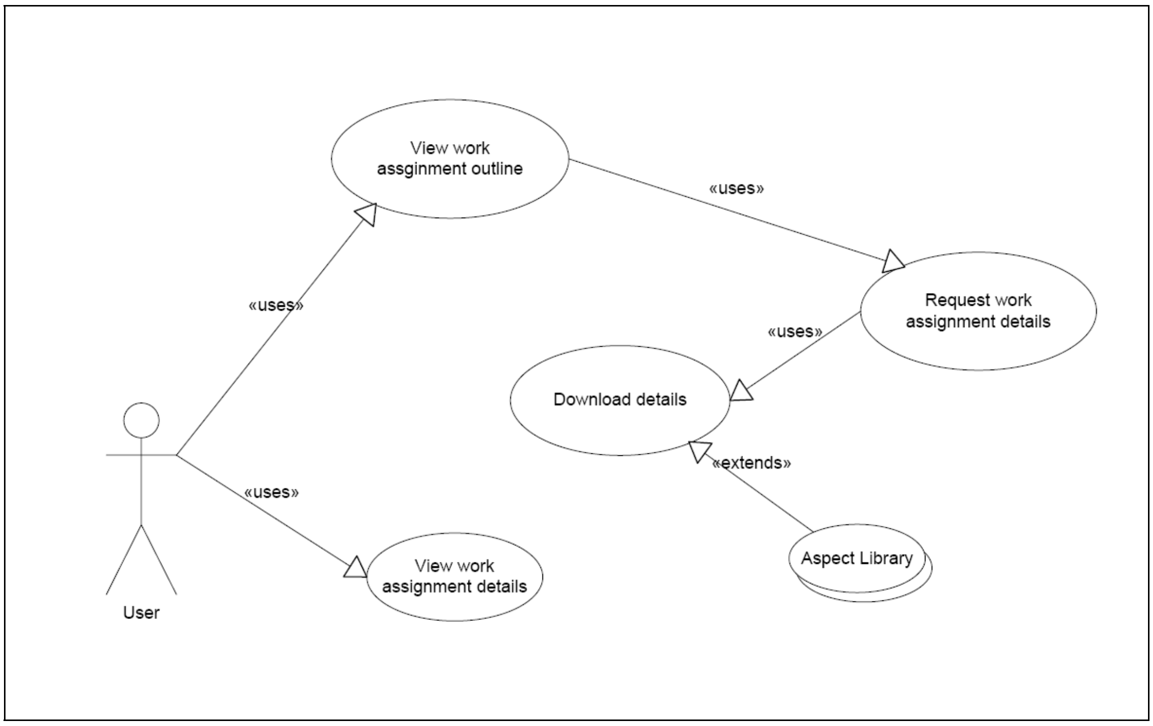
View work assginment outline
«uses»
Request work assignment details
«uses»
«uses»
Download details
«extends»
«uses»
View work assignment details
Aspect Library
User

Figure 4.2 – Elaborated Use Case "Preparing the work"

A textual use case is used to present the details around each use case and particularly the adaptations that take place in the Aspect Library use case. In aspect-oriented programming it is important to define where the aspect should be applied in the code. We have fitted all the use cases into one textual use case.

Table 4.2 – Use Case 1 Preparing the work	
The use case starts when a user selects the view work outline function.	
Basic flow	
1.	User selects view work outline.
2.	System displays the work outline.
3.	User selects task.
4.	System display task summary.
5.	User request work assignment details.
6.	System downloads the details.
7.	System displays the details.
8.	User closes the application.
9.	System shuts down.
Alternative flow	
A1	After Basic flow step 7 the user might return to Basic flow step 4 by pressing "Back".
A2	After Basic flow step 4 the user might return to Basic flow step 2 by pressing "Back".
A3	The User can go to Basic flow step 8 from any step in the Basic flow.
Extension point	
E1	Step 6 in the Basic flow System downloads the details.

Based on the details in the scenario, the use cases in Figure 4.1 and Figure 4.2, and the description provided in Table 4.1, it is possible to define a whole set of adaptations. We have limited ourselves to just three, but in addition to these we might imagine such adaptations as: Switching between network types (WIFI/GPRS), adapting to take advantage of temporary high bandwidth (filling the local repository, pre-fetching, etc), adapting to support a situation where there is no network coverage at all and adapting to a situation with costly network.

In addition to adapting the application to meet other concerns there is also the possibility of using other measures to adapt to the concerns in question. In Use Case 2 we have suggested one way of reducing the memory consumption. The adaptation is a reasonable one that would in most situations reduce the memory consumption slightly, with only a small penalty. It is possible to imagine situations where a more drastic measure would be prudent. If memory is scarce, but battery and bandwidth is available in abundance it would be correct to abolish the local repository altogether and retrieve all requests over the network. Having several levels of adaptations it would be possible to define a large set of adaptations to accommodate a wide range of contextual situations, resulting in greater adaptability.

Although the extension flow of use cases is supposed to point to a specific use case to extend, we have found that in some situations that is not always necessary. For instance when the adaptation we imagine would affect several parts of the system or when it not related to any use case in specific. In these cases we call them generic extensions.

Table 4.3 – Use Case 2 Memory Conservation	
Memory Conservation is performed at the discretion of the adaptation manager and does not relate to any specific extension point.	
Extension flow (generic extension)	
1.	The least relevant entries in the work assignment repository are removed. (The actual change is in the parameters of a purge method that is run on the work assignment repository periodically.)

Table 4.4 – Use Case 3 Power Conservation	
Power Conservation is performed at the discretion of the adaptation manager and does not relate to any specific extension point.	
Extension flow (generic extension)	
1.	A queue is implemented for communication.
2.	Any requests are added to the queue instead of being sent immediately.
3.	The queue is executed at the arrival of priority request, such as a request made by the user. (We make the assumption that different parts of the system make use of the network to synchronize repositories and such.)
4.	The communication unit is switched off between each queue execution.

Table 4.5 – Use Case 4 Low Bandwidth Adaptation	
The Low Bandwidth adaptation is applied to Extension point 1 in Use Case 1 Preparing the work.	
Extension flow (Use Case 1.E1)	
1.	Only Information tagged as “important” is downloaded by the system.

4.2.2 Scenario Description "Measurement"

The Measurement scenario is also a part of The Janitor Scenario from FAMOUS and describes what the janitor does after he has "prepared the work". Quote from [2]:

"Measurement

• *During work, the janitor has to deal with different kinds of temperature sensors. Measurement is performed manually or automatically using Bluetooth. In the later case, various sensor drivers are needed depending on the sensor types. Drivers can be downloaded from the company equipment server.*

• *As the building under inspection is quite large, measurement collection is expected to last at least one hour. In order to reduce battery consumption, the work is performed in a stand-alone mode and measurements are saved locally. However, the network coverage is good and the handheld remains in a connected mode in order to support incoming calls and messages.*

• *In order to perform his work, the janitor has to screw off a protection cabinet on some equipment. As reading instructs becomes cumbersome while he performs this task, the user interface switches to a voice mode.*"

Additionally the MADAM scenario paper [37] suggests adding a requirement to these scenarios; that the janitor when operating in a foreign location would need some form of security to communicate with the company servers.

A use case diagram and a written description of the use cases are also presented in [7]. The use case is depicted in Figure 4.3 and the description in Table 4.2.

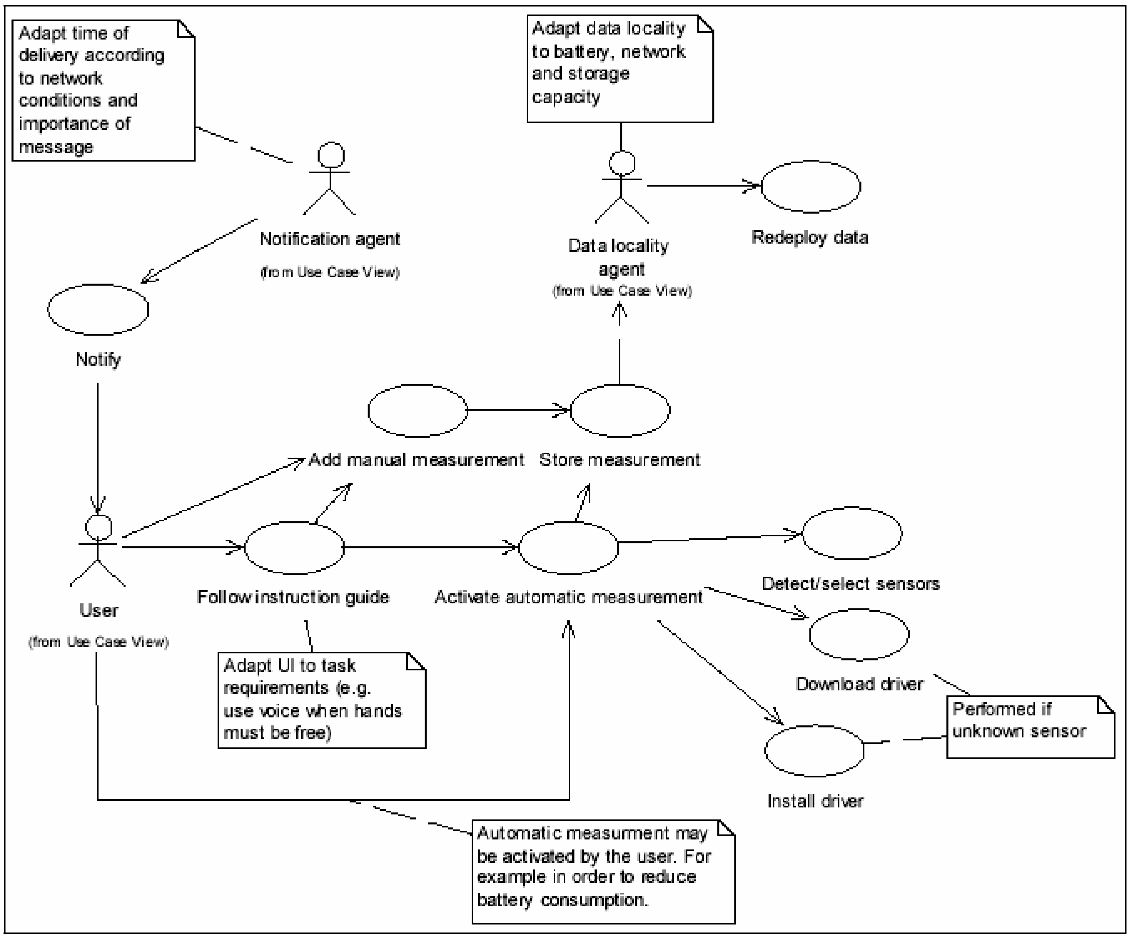
Adapt time of delivery according to network conditions and importance of message
Adapt data locality to battery, network and storage capacity
Notification agent
(from Use Case View)
Data locality agent
(from Use Case View)
Redeploy data
Notify
Add manual measurement
Store measurement
User
(from Use Case View)
Follow instruction guide
Activate automatic measurement
Detect/select sensors
Download driver
Install driver
Performed if unknown sensor
Adapt UI to task requirements (e.g. use voice when hands must be free)
Automatic measurment may be activated by the user. For example in order to reduce battery consumption.

Figure 4.3 – Use Case "Measurement"

Table 4.6 – Use Case "Measurement" from FAMOUS

Use case	**Comment**	**Adaptation to context**
(all use cases)		Long work duration: avoid using the network – however remain connected when possible.
Notify	Incoming messages	Bad network conditions and importance of message: postpone notification
Follow instruction guide	The guide supports each step of the measurement procedure.	No free hands: switch to a voice mode.
Add manual measurement	The instruction guide selects the right application in function of sensor type.	
Store measurement		
Redeploy data	The data locality agent administrates data storage: - Downloading is adapted	Low battery level: do not use network, store data locally.

	to context	Low storage capacity on handheld: copy local data to central storage.
Activate automatic measurement	The instruction guide normally activates the detection of sensors and the measurement. This is performed according to the steps described for measurement procedure.	Low battery level: the user activates automatic measurement.
Detect sensors	Only some of the detected sensors are relevant for the task to be performed. The selection is performed automatically.	
Download driver	Performed automatically	No network coverage: postpone driver downloading.
Install driver	Performed automatically	Unknown sensor type: the driver is bound to the application.

Doing the same kind of transformation as we did to scenario 1 yields the Use Case depicted in Figure 4.4. A textual description of each use case can be found in Tables 4.7 – 4.11. The notify use case is skipped in its entirety. The install driver use case has been integrated with the download driver use case. Follow instruction guide use case has been integrated in the automatic measurement use case.

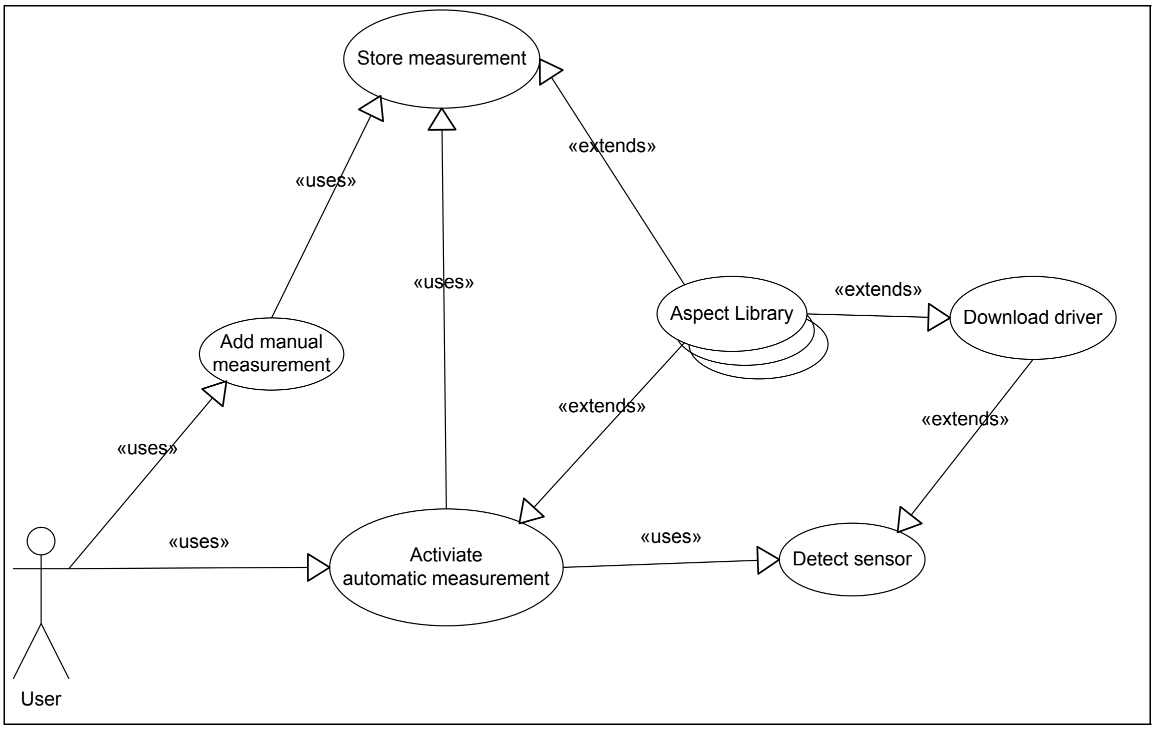

Figure 4.4 – Elaborated Use Case "Measurement"

Table 4.7 – Use Case 5 Automatic Measurement	
The use case starts when a user selects start automatic measurement.	
Basic flow	
1.	User selects “Start automatic measurement”
2.	System performs Use Case 8 Detect Sensor.
3.	System presents result to user.
4.	User confirms storing of measurement.
5.	System performs Use Case 7 Store Measurement.
6.	System returns to Measurement menu.
Alternative flow	
A1	Steps 2-5 performed repeatedly until Use Case 8 reports no new sensors.
A2	In step 4 User can choose abort. This causes the system to jump to step 6.
Extension point	
E1	Step 3 in the Basic flow: System presents the result to user.
E2	Step 4 in the Basic flow: User confirms storing of measurement.

Table 4.8 – Use Case 6 Manual Measurement	
The use case starts when a user selects manual measurement.	
Basic flow	
1.	User selects “Add Manual Measurement”
2.	System displays fields for sensor/room, measurement and comment
3.	User fills in fields and press store data.
4.	System performs Use Case 7 Store Measurement.
5.	System returns to Measurement menu.

Table 4.9 – Use Case 7 Store Measurement	
The use case is called from Use Case 5 Automatic Measurement or Use Case 6 Manual Measurement.	
Basic flow	
1.	System connects to remote repository.
2.	System sends data over network.
3.	System receives confirmation that data has been stored at remote repository.
Alternative flow	
A1	If system is already connected to remote repository, step 1 is skipped.
Extension point	
E1	Whenever using the network

Table 4.10 – Use Case 8 Detect Sensor	
The use case is called from Use Case 5 Automatic Measurement	
Basic flow	
1.	System scans for Bluetooth devices.
2.	System finds a new Bluetooth device.
3.	System retrieves a measurement from the Bluetooth device.
4.	System returns the result.
Alternative flow	
A1	In step 2 if no new Bluetooth device is found, system returns with notification

	of user.
Extension point	
E1	Step 2 System finds a new Bluetooth device.

Table 4.11 – Use Case 9 Download Driver	
The use case starts when a user wants to preload necessary drivers before performing a measurement. The use case also extends Use Case 8 Detect Sensor.	
Basic flow	
1.	User selects "Preload drivers"
2.	System connects to the company equipment server.
3.	System retrieves a list of relevant device drivers from the equipment server.
4.	System presents the list to the user.
5.	User selects the device driver to download.
6.	System downloads the device driver from the equipment server.
7.	System adds the driver to the local driver repository.
Extension flow (Use Case 8.E1)	
1.	System retrieves driver corresponding to the detected device from the local driver repository.
2.	System installs driver in application.
Alternate flow	
A1	In Extension flow step 1 If the device driver is not found in the local driver repository, the system downloads it from the company equipment server.
Extension point	
E1	Whenever using the network

In addition to the textual description of the use cases we provide a description of the adaptations we have thought up. Once again we only present a small subset of possible adaptations. Some are similar to the ones presented in Section 4.2.2 but we nonetheless choose to represent them anew as there are some differences in what way they adapt the application.

Table 4.12 – Use Case 10 Memory Conservation	
Memory Conservation is performed at the discretion of the adaptation manager and does not relate to any specific Extension Point defined elsewhere in the system.	
Extension flow (generic extension)	
1.	The least relevant drivers in the local driver repository are removed.
2.	The local storage holding temporary measurement data are forced to remote storage.

Table 4.13 – Use Case 11 Power Conservation	
Power Conservation is performed at the discretion of the adaptation manager and does not relate to any specific Extension Point defined elsewhere in the system.	
Extension flow (generic extension)	
1.	A queue is implemented for communication.
2.	Any requests are added to the queue instead of being sent immediately.
3.	The queue is executed at the arrival of priority request, such as a request made by the user. (We make the assumption that different parts of the system make use of the network to synchronize repositories and such.)

4.	The communication unit is switched off between each queue execution.

Table 4.14 – Use Case 12 Hands-free	
The use case extends Use Case 5 Automatic Measurement.	
Extension flow (Use Case 5.E1 & E2)	
1.	System presents the results to the user with an audio representation of the measurement.
2.	User confirms the storing of the measurement using a vocal command.

Table 4.15 – Use Case 13 Security	
The use case extends all Use Cases using the network. Currently Use Case 7 Store Measurement and Use Case 9 Download driver.	
Extension flow (Use Case 7.E1 9.E1)	
1.	System switches to secure socket for external communication.

4.3 Implementation Tools

The AOP framework that provides best support for dynamic adaptation is PROSE. It is the only framework that provides run-time weaving of aspects through the technique of just-in-time (JIT) compiler weaving which is described in more detail in Section 3.1.

As we are dealing with adaptive systems the ability to dynamically weave aspects in and out is a minimum requirement. Several of the frameworks presented in 3.1 support this (Aspectwerkz, JBoss AOP, PROSE, and JAC). PROSE provides the best support for dynamism with its JIT compiler weaving and is the most eligible choice. However, after having spent the better part of a week trying to configure it for use on Eclipse under Windows XP, and another week for Eclipse under Debian Linux, we have come to the conclusion that we might be better served with a more user-friendly framework. We have also found the documentation lacking as it has not been updated since PROSE version 1.0.2 and no longer relates to the current executables. Another issue is the use of the IBM research virtual machine “Jalapeno”, which does not seem to like either of the platforms we tried.

After discarding PROSE we decided to go with JBoss AOP as it provides load-time weaving and has a larger support organization that provides frequent updates and verbose documentation while being released under the LGPL. JBoss integrates flawlessly with the Eclipse IDE. Using the update-mechanisms in Eclipse ensures a plug-in that is built for the right platform, and all run-time parameters to the JVM are defined automatically. JBoss AOP’s Eclipse plug-in also provides among other things wizards for creating new classes, functionality for auto-generating xml-files for binding and a debugging tool.

As IDE we chose Eclipse. A good IDE is necessary for large AOSD projects where many bindings and crosscutting aspects might create an unmanageable situation. The Eclipse foundation also maintains and develops the AspectJ language and many of the other AOP frameworks provide plug-ins for Eclipse. Eclipse is a central tool in the AOSD community.

We created a sample application where we used JBoss AOP to weave some aspects.

In order to use aspect-adaptation it is necessary to have some code to advice. Due to the time constraints we decided against creating a working application, and instead focus on creating a working evaluation and change mechanism. This lead to a situation where we have no code to weave the aspects into. So no actual weaving takes place in our prototypes.

4.4 Design and Implementation

Two separate implementations were made. Prototype 1 is based on the Preparing the work scenario described in Section 4.2., while Prototype 2 is based on the Measurement scenario, also presented in Section 4.2.2.

Both prototypes interact with the context simulator which was developed as a part of the FAMOUS project to mimic a context environment. The context simulator generates context events which our prototypes subscribe to by implementing the interface ContextChangeListener. The Context simulator is described in detail in [5].

4.4.1 Prototype 1

In the first prototype we have focused on creating a mechanism for aspect adaptation, in contrast to component framework recomposition that is in use in the current FAMOUS implementation. We started with a very primitive model. We assigned a set of variables to the application. Memory use, power use and bandwidth use were chosen initially as these are aspects that we wanted to adapt the application to.

The design is based on the use case in Figure 4.2. FAMOUS had already made a design based on the use case in Figure 4.1. We based our design on theirs and added the changes from our use case.

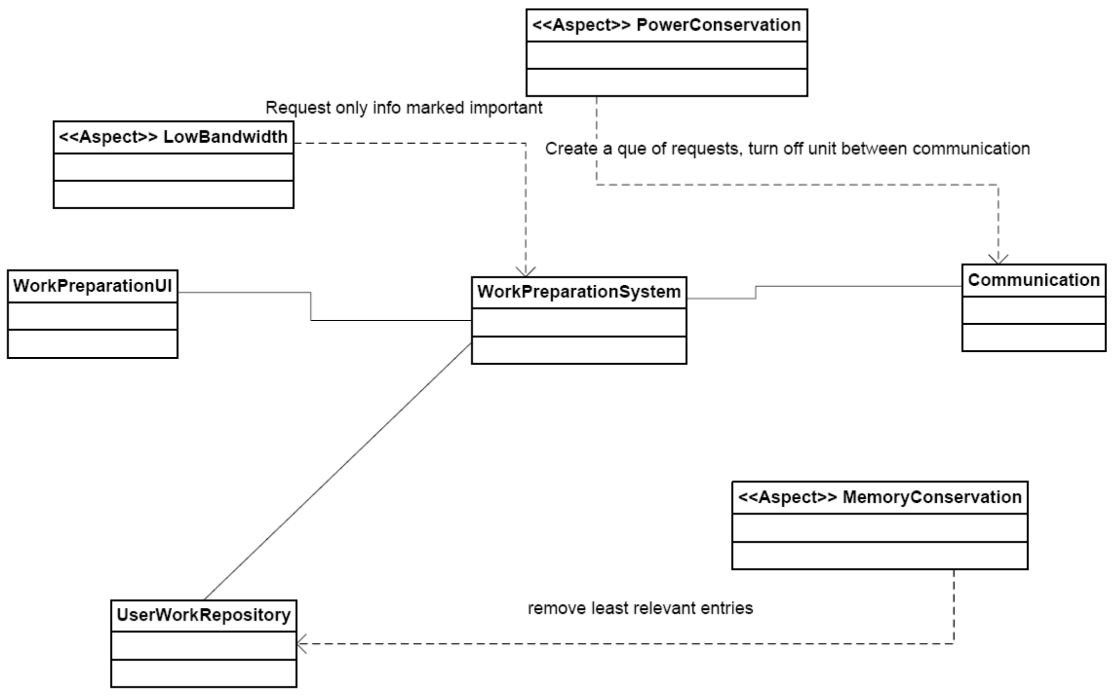
<<Aspect>> PowerConservation
Request only info marked important
<<Aspect>> LowBandwidth
Create a que of requests, turn off unit between communication
WorkPreparationUI
WorkPreparationSystem
Communication
<<Aspect>> MemoryConservation
remove least relevant entries
UserWorkRepository

Figure 4.5 – Class Diagram Prototype 1 - Application and Aspects

Figure 4.5 is a class diagram of the application part of the system. The application classes, WorkPreparationUI, WorkPreparationSystem, Communication and UserWorkRepository are only implemented as shell classes. No functionality has been coded into these classes. Similarly, the aspects contain no actual code to adapt these empty classes. The actual adaptation does not take place. Only the adaptation system and evaluation functions are implemented. Furthermore Prototype 1 contains no actual aspect code, no actual weaving of aspects takes place.

We use the context simulator created as a part of the FAMOUS project [5] to vary and get data on the context. The context simulator gets its initial context from an XML-file that it reads. It contains a GUI where one can change the values of the different context properties. This is the same approach as is used by the FAMOUS prototype v1 [7].

Our prototype keeps a model of the application. This model is the basis for deciding how to adapt the application best to the current context. The actual reasoning behind the evaluation of the application is very simple. A set of properties about the application is defined. This set is extendable but initially we have chosen the application's memory use, the current bandwidth need of the application and the current power use of the application. These properties are assigned a value, called the base value. There are a set of aspects which affect the application. Aspects can be added but the initial set is memory-conserving aspect, power-conserving aspect and bandwidth-conserving aspect.

Since our set of properties of the application is so small we have added a third property to represent changes which reduce the functionality of the application. We call this property a utility penalty and it would represent such properties as response time or latency, lack of functionality in the application and other effects which we have decided not to model in this prototype. The effect of the individual aspects and the complete list of the different configurations and their properties are listed in Tables 4.16 and 4.17.

Table 4.16 – Prototype 1: The Application and Aspect Properties

	Memory Use	Power Use	Bandwidth needed	Utility penalty
Base	30	30	30	0
mca	-20	+10	+10	0
bca	+10	+10	-10	-5 %
pca	0	-10	0	-5 %
mca = Memory Conserving Aspect, bca = Bandwidth Conserving Aspect, pca = Power Conserving Aspect				

Table 4.17 – Prototype 1: The Properties of all Configurations

	Memory Use	Power Use	Bandwidth needed	Utility penalty
Base	30	30	30	0
Base+mca	10	40	40	0
Base+mca+pca	10	30	40	-5 %
Base+mca+bca	20	50	30	-5 %
Base+mca+pca+bca	20	40	30	-10 %
Base+bca	40	40	20	-5 %
Base+pca	30	20	30	-5 %
Base+pca+bca	40	30	20	-10 %
mca = Memory Conserving Aspect, bca = Bandwidth Conserving Aspect, pca = Power Conserving Aspect				

These properties are the tools with which we can adapt the application to the context. The context properties we have decided to collect are memory available, the bandwidth of the current connection and the state of the battery. When running our prototype we use the context simulator developed as a part of the FAMOUS project to vary the context. Our prototype will respond to the context changes by adapting the application if it finds that beneficial. We have not provided a log of the running of Prototype 1 as a running of Prototype 2 would provide a more interesting reading.

The evaluation system, located in the Evaluator class, takes input of a hashtable of the applications properties and a hashtable of the context. In the case of the memory evaluation, the system gives max utility (100) if the amount of memory is 10 units larger than the amount of memory used by the application. It declines proportionately down to where available and used is the same amount and returns 0 utility if the amount of available memory is lower than the amount required. Similar evaluations are carried out for battery level and bandwidth. Finally the three values are summed up and the utility penalty is subtracted.

4.3.2 Prototype 2

In Prototype 2 we used a component based model of the application. All classes of the application subclass from the Component class, which holds a set of properties for that component. The evaluation functions sums up these properties along with the effects of the aspects and the evaluation is similar to that of Prototype 1.

The classes of the application together with the aspects and their measures to adapt the application are depicted in Figure 4.6.

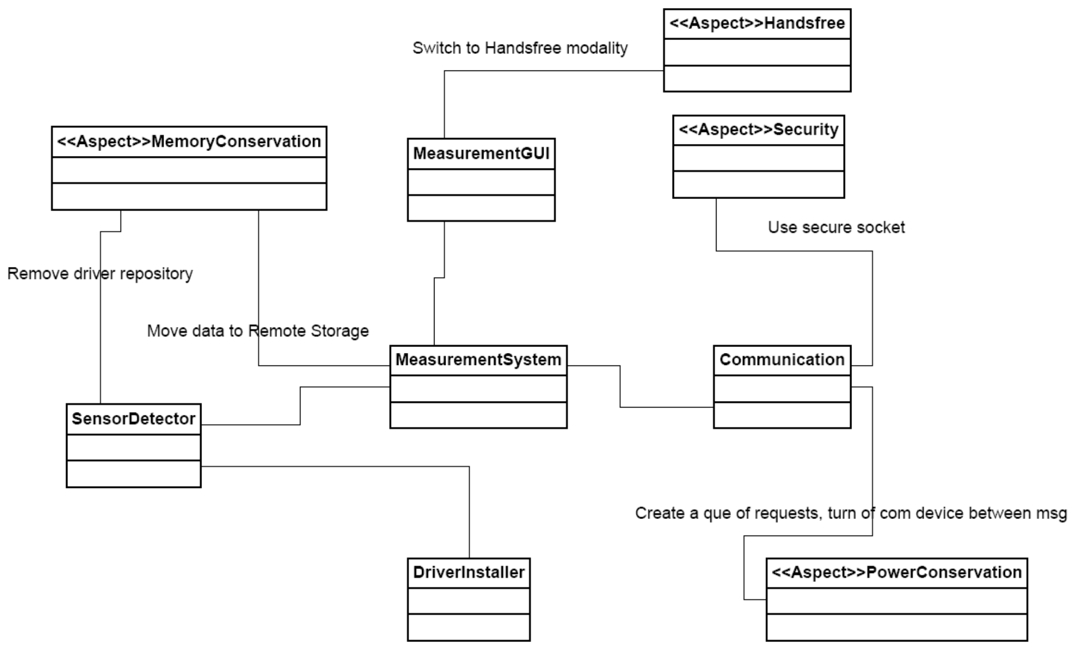
Switch to Handsfree modality
<<Aspect>>Handsfree
<<Aspect>>MemoryConservation
MeasurementGUI
<<Aspect>>Security
Use secure socket
Remove driver repository
Move data to Remote Storage
MeasurementSystem
Communication
SensorDetector
Create a que of requests, turn of com device between msg
DriverInstaller
<<Aspect>>PowerConservation

Figure 4.6 – Class Diagram Prototype 2 - Application and Aspects

In Figure 4.7 we display the class diagram of the complete system. The Application and the aspects are located in one package, and the adaptation system is located in another.

There are two inheritable classes in our Prototype. The Component class is a concrete super-class that all components of the application must inherit from. The Component class holds a list of properties where the sub-class must specify its memory consumption and latency contribution etc. AspectProperties is an inferface that must be implemented by all aspects that are a part of the adaptation system. (There might be aspects in the application that is not a part of the adaptation system.)

Two data classes have been defined. The Effect class holds information on the effects of an aspect. Aspects might affect several components. The Config class holds the configuration as an array of ones and zeros; in addition to the utility that the configuration gave at the last evaluation.

Three classes make out the core of the adaptation system. The Evaluator takes the context and a configuration as input and returns the utility of that configuration in the given context. The AspectLibrary contains a reference to all aspects. Finally the AspectAdaptationSystem is where the bulk of the work is done. It implements the interfaces ContextChangeListener and SimulationManager in order to, respectively, listen to changes from the context simulator and manage the context simulator. The context simulator is a part of the FAMOUS prototype which we use to simulate context.

The aspects are added with a recursive greed algorithm which first adds the aspect with the greatest improvement in utility and then the second greatest and so on until no aspect can improve utility.

The rationale behind having a greed algorithm to evaluate the different configurations is that the number of combinations will increase at a polynomic rate. The number of combinations can be expressed with 2^n where n is the number of aspects. With only 16 possible configurations in this prototype it is not strictly necessary, but one does not need many aspects before a complete evaluation of all configurations become very time-consuming.

Whether this algorithm actually finds the optimal solution will depend, we believe, on the values given to the different aspects. The discussion of whether it finds the optimal solution and how far off the optimal its results are is beyond the scope of this project.

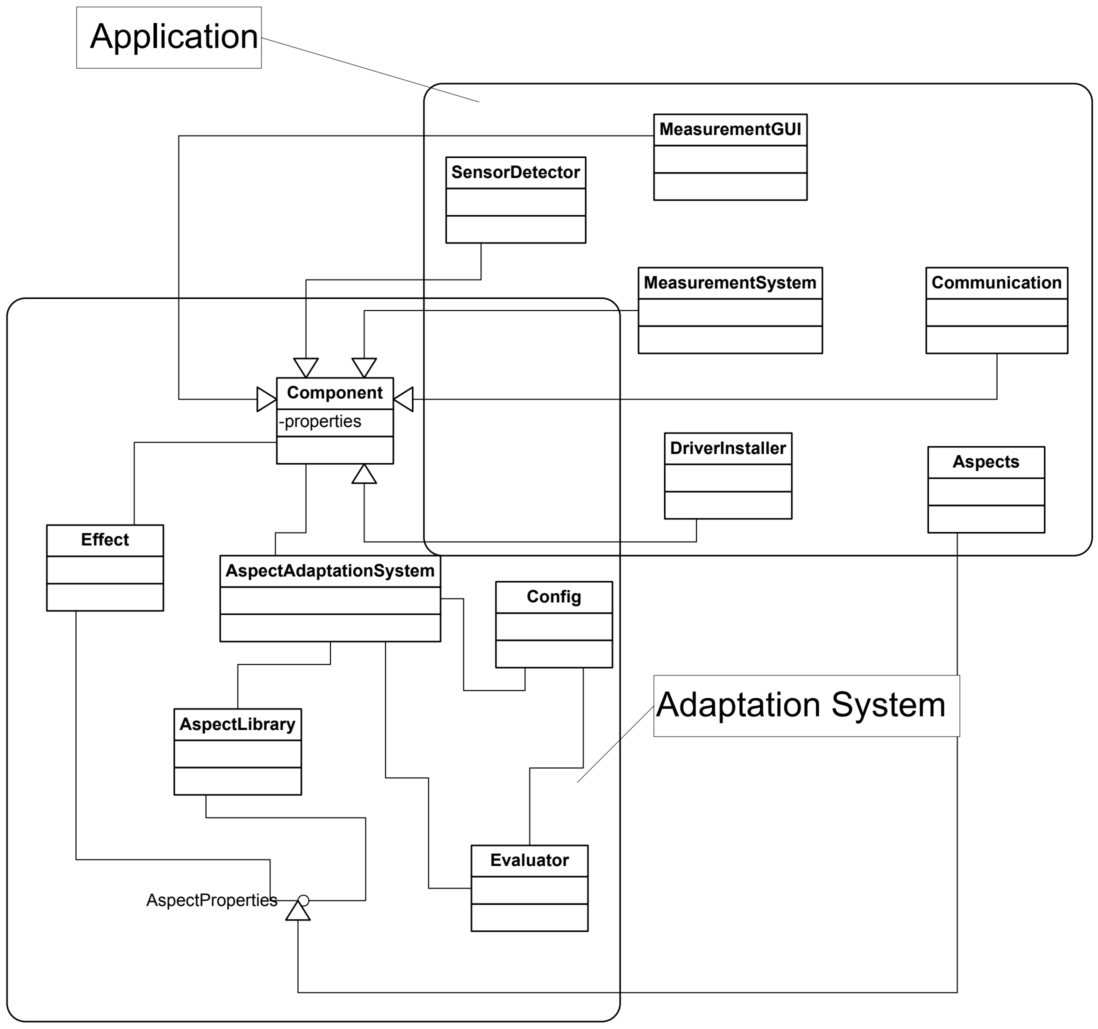

Figure 4.7 – Class Diagram Prototype 2 - Application and Adaptation system

4.5 Test-Running the Prototype

We ran several tests to see how well our prototype adapted to the context. We only provide a report from the running of Prototype 2 as that is the most developed prototype. We created a usage scenario which contains a set of context changes in an ordered manner. The usage scenario is loosely based on the scenario “Measurement” presented in Section 4.2.2.

In the initial state the application has adequate memory, bandwidth is no concern, no security or hands-free is required and the latency tolerated variable is set to 20. The scenario progresses as follows: The janitor arrives at the customer location; there is an open WIFI network available, but as the application transmits and receives sensitive data a secure communication is required. This is simulated by upping the security required variable from 0 to 100. The measurements take place and take some time. The intensive use of Bluetooth rapidly drains the battery; power available drops from 90 to 30. The janitor has to unscrew a cabinet, requiring hands-free operation; hands-free goes from 0 to 100. The janitor is finished and is leaving the customer area, thus eliminating the need for security; security required drops from 100 to 0. The janitor goes to his car so he continues to use hands-free operation. The janitor starts a memory intensive GPS map/pathfinder application on his PDA to find his way back to the office; Memory drops from 90 to 50. The janitor remembers to plug in the PDA to the car’s power outlet; Power available rise from 30 to 90.

Table 4.18 shows how the prototype responds to these context changes. A summary, which is somewhat more readable, is provided in Table 4.19. The full trace can be found in Appendix C.

Table 4.18 – Trace from the Test-run of Prototype 2

	mem	nbw	pow	haf	lat	sec	Config	U	Config	U	Config	U	Config	U	Config	U
Initial State	90	70	90	0	20	0	A0000	100								
Arriving at Customer	90	70	90	0	20	**100**	A0000	83	A1000	83	A0100	83	**A0010**	**100**	A0001	66
Recursive call							A1010	94	A0110	94	A0011	77				
Power dropping	90	70	**30**	0	20	100	A0010	83	A1000	66	A0100	75	A0001	50		
Recursive call							A1010	77	**A0110**	**86**	A0011	61				
Recursive call							A1110	83	A0111	66						
Requiring hands-free	90	70	30	**100**	20	100	A0110	69	A1000	50	A0100	58	A0010	66	A0001	66
Recursive call							A1010	61	A0011	77						
Recursive call							A1011	75	**A0111**	**83**						
Recursive call							A1111	81								
Leaving customer	90	70	30	100	20	**0**	A0111	83	A1000	66	A0100	75	A0010	66	A0001	83
Recursive call							A1001	80	**A0101**	**88**	A0011	77				
Recursive call							A1101	84	A0111	83						
Starting new App.	**50**	70	30	100	20	0	**A0101**	**71**	A1000	66	A0100	58	A0010	50	A00001	66
Recursive call							A1100	71	A1010	61	A1001	71				
Recursive call							A1110	58	A1101	67						
Power in car	50	70	**100**	100	20	0	A0101	80	A1000	83	A0100	66	A0010	66	A0001	83
Recursive call							A1100	80	A1010	77	**A1001**	**88**				
Recursive call							A1101	76	A1011	75						

mem = Memory Available, nbw = Network Bandwidth, pow = Power Available, haf = Hands-free Required, lat = Latency Tolerated, sec = Security Required, Config is the configuration and the four digits represent the four aspects, in the order memory, power, security, hands-free, and 1 represents active and 0 represents inactive. The column marked U represent the configuration’s utility. The Changes in the context and the chosen configuration is marked in Bold.

Table 4.19 – Summary of Trace from Test-run of Prototype 2	
Initial State	No aspects active
Arriving at Customer	Add security aspect
Power dropping	Add power conserving aspect
Requiring hands-free	Add hands-free aspect
Leaving customer area	Remove security aspect
Memory Intensive App	No change
Power available in car	Remove power conserving aspect and add memory conserving aspect

The prototype holds up well and performs within expected parameters. As we see from Table 4.18 there are three configurations which yield the same utility when the Memory Intensive application is started. The system prefers not make changes when no improvement can be found. The memory conserving aspect is needed in that situation. The combination of the memory and the power conserving aspects increase the latency beyond the tolerance. With a higher latency tolerance, the aspect would have been weaved in at that point. As soon as the power conserving aspect is no longer needed, the memory conserving aspect is immediately added.

This test indicates that a relatively primitive model and evaluation scheme can successfully adapt to many situations.

5 Results

We present some new ideas. Using a simpler meta-model to model aspects for fine-grained adaptations that in turn can be combined with component evaluation and more coarse-grained adaptations. Weaving changes to the model, evaluation function and utility function into the model while weaving the actual changes into the application. Using a greed algorithm for adaptations to create a best effort adaptation that is not always optimal in situations where a complete evaluation of all configurations is impossible or very time-consuming.

We created a prototype with a simple evaluation model. We tested our simple model and it performed as expected. The FAMOUS evaluation model is able to represent structure, distribution, variability and property specification.

The FAMOUS model covers structure by modelling components that are connected through bidirectional ports that define services that components can provide or use. Our model represents structure less formally by representing components that consume or provide properties from and to a pool of resources.

The FAMOUS model allows components to be distributed on different nodes. Our model does not support distribution.

The FAMOUS model represents variability by allowing the planner to choose several different compositions of components. Different components can implement the same role. Our model allows variability to be modelled as set of aspects which are either active or inactive.

The FAMOUS model evaluates how well the different compositions use the resources provided by the underlying system to provide services consumed by the user. Our model uses the same approach.

The FAMOUS prototype requires that an evaluation function exist for component and that a utility function exist for each possible configuration. Our model uses an evaluation function for each property and a single utility function. We believe our model will scale well as more adaptations are added. We realize that the model is a rather rough approximation to the application but we believe that our prototype show that it performs acceptably.

Our prototypes only contain one advice per aspect. It would be possible to imagine a whole set of advice code that can be added to improve a particular property. All these measures could be contained in one aspect. Our suggested efforts to reduce memory consumption are not very drastic and would not lead to any significant loss of functionality. It is possible to imagine a situation were a more dramatic reduction in memory use would become beneficial.

We believe variability is more easily implemented with aspects than with components. We believe that using aspect-oriented software development to create applications lead to more natural variability. Fine-grained variability can be added by several small steps. Using aspect-oriented Analysis on the application in both the design and implementation phase of the project would easily identify potential adaptations.

6 Evaluation

In Section 2.2 we formulated five research questions. We here give a brief summary of what we have found to be the answers to these questions.

1) Are aspect-oriented programming mechanisms for run-time adaptation better suited than component framework reconfiguration?
2) Are aspect-oriented programming mechanisms for run-time adaptation better suited at adapting fine-grained adaptations than component frameworks?
3) Are aspect-oriented programming mechanisms for run-time adaptation better suited at adapting cross-cutting concerns than component frameworks? Are crosscutting concerns commonly reconfigured in adaptive systems?

Aspect-oriented mechanisms for adaptation operate at a more fine-grained level and are ideal for smaller changes. Component reconfiguration is the natural choice for changing the architecture of the application and doing other large scale changes. The two methods complement each other and can be combined. Distribution of components among several nodes is not easy to implement as aspects and can better be achieved using component reconfiguration. For making smaller, crosscutting adaptations to running components, like changing buffer sizes or adjusting variables aspect adaptation is better suited than component reconfiguration.

We believe crosscutting concerns are quite commonly adapted. The PROSE team has concluded this as well: “We have encountered a number of scenarios where adaptations cut across a mobile system that needs to be adapted.” (Chapter 31 in [10]). We mention a couple of common examples for when the adaptation is crosscutting in nature: Whenever a component is removed or exchanged for another component, all objects that refer to that component would have to get their references updated. When the protocol used is exchanged for another all objects that communicate over the network would have to have their method calls changed. This is quite easily expressed in a single statement in AOP.

4) Does aspect-oriented software development lead to less and simpler meta-information about the (reconfiguration of) the application?

Our model is certainly simpler than the one employed by the FAMOUS project. This might be because aspects are defined to affect a specific component, whereas components can be interconnected in several different ways. However, our model does not take distribution into account and is simpler because we have chosen to have generic evaluation and utility functions. Our work suggest that aspects require less meta-information, but this may simply be because we have chose to use a more simple model. So the answer to the question remains inconclusive.

5) Is the current FAMOUS utility evaluation technique for component reconfiguration applicable for aspect adaptation?

We suggest three approaches for extending the current FAMOUS evaluation method to support aspects. Our prototype is based on one of these approaches suggested. The actual combination of the FAMOUS evaluation system with ours is not implemented in our prototype but we believe that it is possible.

Is aspect-oriented software development a suitable development technique for development of adaptive systems?

We believe the technology of AOP/AOSD have a lot to offer to the field of adaptive systems. We see several potential benefits of AOSD. The techniques of looking at the application with different aspects/viewpoints in mind and seeing what adaptations the applications can accommodate. AOSD is a helpful tool in creating variability engineered applications. The aspect weaving process for implementing the actual adaptation shows promise. An aspect-oriented change mechanism is especially suited to adapt fine-grained and cross-cutting concerns. We believe aspect-oriented software development is a suitable development technique for development of adaptive systems. The benefits of using AOSD will only increase as the field matures.

7 Related Work

Finding fields of problems where aspect-orientation is especially suited is an ongoing project in many parts of the AOSD community. It seems that adaptive systems is one such field that might benefit from AOSD.

A team of researchers from Michigan State University has done something very similar to what is explored in this paper [36]. They wanted to create a system for dynamic adaptation and chose an aspect-oriented approach to do this. They based their work on AspectJ. They chose not to use any aspect framework's support for dynamic weaving, but rather created their own wrappers that makes call to an adaptation kernel that decides whether new code should be added to the system.

The group behind PROSE has been using PROSE to create an adaptive system. Their efforts are described in Chapter 31 of [10]. They have used their system to create location specific behavior of robots using Lego's Robotics Invention System and iPAQ PDAs.

The Politecnico di Torino have used PROSE to create adaptive middleware using aspect weaving as a change mechanism. The JADDA Framework (Java Adaptive component for Dynamic Distributed Architectures) is based on aspect weaving, xADL description languages and generic middleware. The project is described in [23].

The PROSE group is also involved in several other projects that use AOSD to create adaptive systems.

8 Conclusions and Further Work

Here we present our conclusions and avenues of further work.

8.1 Conclusion

We have created a prototype of an evaluation system for an adaptive system that uses aspect weaving to perform adaptations. We suggest that AOSD can be very beneficial in the development of adaptive system. Aspect weaving is a very good change mechanism for fine-grained changes and an excellent change mechanism for crosscutting changes. The process of variability engineering of applications for adaptive platforms can be made easier using AOSD.

8.2 Further Work

Getting some empirical data on how well aspect-weaving performs compared to component reconfiguration is necessary in order to make informed decisions on what system is better suited to which conditions. The PROSE team claims that normal running of an application with no weaving using a total hook policy would not decrease performance more than 10%. A potential follow-up project could use benchmarking to verify that these results also hold for adaptive systems and compare how well aspect-weaving at runtime compares to component reconfiguration.

An interesting ability of PROSE is that it blocks joinpoints and all objects who try to use them until the weaving process is completed. It could be interesting to investigate if this mechanism could eliminate the necessity of using suspend protocols to get applications to a safe state before reconfiguration as is used in FAMOUS today.

It would be interesting to see how well our model scaled as the number of aspects increase. Few changes to the system would be required to test this. By adding a number of aspects with different properties and maybe adding some properties the model takes into account.

Exploring how well a greed algorithm for evaluating configurations performs could be an interesting follow up. Checking whether the algorithm finds the optimal solutions and on what parameters finding the optimal solution is dependent on would be interesting knowledge.

Appendix A – Reference

[1] The FAMOUS website: www.famous-project.net Accessed 09.01.05.

[2] Odd-Wiking Rahlff, E. Stav, F. Vraalesen, J. Floch. **"Context model"** Famous deliverable D1.1 – v.1.0. SINTEF report STF90 A04043.

[3] MADAM – Mobility and Adaptation enabling Middleware – project; **"Annex I – Description of work"**

[4] FAMOUS – Framework for Adaptive Mobile and Ubiquitous Services, SINTEF Telecom and informatics 2003-2006, Strategic Research Programme.

[5] Jacqueline Floch, Erik G. Nilson: **"Context Simulator"** Famous deliverable D1.2 – v1. SINTEF report STF90 A04044.

[6] Erlend Stav, Svein Hallsteinsen: **"Definition of adaptive architecture"** Famous deliverable D3.1 – v1. SINTEF report STF90 A04046.

[7] Jacqueline Floch, Erlend Stav, Erik G. Nilsson: **"Prototype Service"** Famous deliverable D3.3 – v1. SINTEF report STF90 A04047.

[8] Svein Hallsteinsen, Erlend Stav: **"Experience with prototype service.** Famous deliverable D3.4 – v1. SINTEF report STF90 A04048.

[9] Ramnivas Laddad: **"AspectJ in Action"** 2003 Manning Publications Co ISBN 1-930110-93-6

[10] Robert E. Filman, Tzilla Elrad, Siobhán Clarke, Mehmet Aksit: **"Aspect-oriented Software Development"** 2005 Addison Wesley. Pearson Education. ISBN 0-321-21976-7. The book is a collection of 31 articles. I refer only to the chapter of the book. For details of the book and each article, please see: http://home.comcast.net/~filman2/aosd-book.html

[11] Elrad et al: **"Aspect-oriented programming: Introduction"** (ACM 2001)

[12] Gregor Kiczales et al: **"Aspect-oriented programming"** Xerox Palo Alto Research Center 1997

[13] Hallsteinsen, Floch and Stav: **"A Middleware Centric Approach to Building Self-Adapting Systems"** (2004, SINTEF ICT, 7465 Trondheim, Norway)

[14] Szyperski: "Component Software: Beyond Object-Oriented Programming", 1997 Addison Wesley, ISBN 0-201-74572-0

[15] Halsteinsen, Stav and Floch: **"Self-Adaptation for Everyday Systems"** (WOSS 2004)

[16] The aspect-oriented software development community and conference website: aosd.net/ Accessed 13.01.05

[17] The AOP alliance website: aopalliance.sourceforge.net

[18] The Eclipse foundation's AspectJ website: eclipse.org/aspectj/ Accessed 01.03.05.

[19] The PROSE project website: prose.ethz.ch Accessed 17.01.05

[20] Andrei Popovici: **"PROSE user manual"** version 1.0.2. Swiss Federal Institute of Technology Zürich. Available at prose.ethz.ch/webthings/manual.pdf

[21] Andrei Popovici, Thomas Gross and Gustavo Alonso**: "Dynamic Weaving for Aspect-Oriented Programming"** (AOSD 2002)

[22] Andrei Popovici, Gustavo Alonso and Thomas Gross: **"Just-In-Time Aspects: Efficient Dynamic Weaving for Java"** (AOSD 2003)

[23] Paolo Falcarin and Gustavo Alonso: **"Software Architecture Evolution through Dynamic AOP"** (EWSA 2004/ICSE 2004)

[24] The JBoss AOP website: www.jboss.org/products/aop Accessed 04.03.05

[25] The JBoss AOP 1.1 User Guide. Available at: http://docs.jboss.org/aop/1.1/aspect-framework/userguide/en/html/index.html

[26] The JBoss AOP 1.1 Reference Guide. Available at: http://docs.jboss.org/aop/1.1/aspect-framework/reference/en/html/index.html

[27] B. Burke, A. Chau, M. Fleury, A. Brock, A. Godwin, and H. Gliebe: **"JBoss Aspect Oriented Programming"** (The JBoss Group 2003)

[28] The Aspectwerkz project website: aspectwerkz.codehaus.org Accessed 04.03.05

[29] The JAC project website: jac.objectweb.org

[30] Renaud Pawlak: **"Interaction-based Aspect Oriented Programming for Constructing Multiple Concern Applications"** French National Institute of Arts and Crafts (CNAM), CEDRIC laboratory, Paris, France. 2002.

[31] Ruzanna Chitchyan, Ian Sommerville: **"Comparing Dynamic AO Systems"** (AOSD 2004)

[32] M.E. Fayad, Anita Ranganath: **"Modeling Aspects using Software Stability and UML"**

[33] Dominik Stein, Stefan Hanenberg, and Rainer Unland: **"Designing Aspect-Oriented Crosscutting in UML"**

[34] Renaud Pawlak et al: **“A UML Notation for Aspect-Oriented Software Design”**

[35] Ivar Jacobson and Pan-Wei Ng: **“Aspect-oriented software development with use cases”** 2004 Addison Wesley. Pearson Education. ISBN 0-321-26888-1

[36] Z. Yang, B. H. C. Chang, R. E. K. Stirewalt, S. M. Sadjadi, P. K. McKinley: **“An Aspect-Oriented Approach to Dynamic Adaptation”** Michigan State University. (WOSS 2002)

[37] Akis Chimaris, George Papadopoulos, Nearchos Paspallis and Zacharias Abraham: **“Adaptation Scenarios and Reference Requirements for Service Adaptation”** University of Cyprus, MADAM project Deliverable 1.1 (v3).

[38] Victor R. Basili: **“The Experimental Paradigm in Software Engineering.”** 1992. Springer Verlag LNCS 706.

[39] Marvin V. Zelkowitz and Dolores R. Wallace: **“Experimental Models for Validating Technology”** *IEEE Computer*, 31(5), May 1998.

[40] Wikipedia, Online dictionary: www.wikipedia.org Accessed repeatedly.

Appendix B – Glossary

Some of these definitions come from the books or articles listed in the Reference. Others were collected from Wikipedia [40].

Advice	From AOSD: Advice is the code that is applied at a pointcut.
AOP	Aspect-Oriented Programming
AOSD	Aspect-Oriented Software Development. The difference between AOP and AOSD: When we talk about AOP in this paper we usually refer to the frameworks and the programming languages. When we refer to AOSD we think of the whole process of aspect-oriented software development, that is, from requirements gathering through modelling, design, implementation, testing and deployment. Since AOP/AOSD is a relatively fresh field there are still different usages of the terms.
API	Application Programmer's Interface
Aspect	An aspect is the basic unit in aspect-oriented programming. An aspect can contain a pointcut which refers to one or more joinpoints, the advice which is to be applied at the pointcut and introduction. In addition aspects can have fields and methods. Aspects can also be instantiated in some AOP system, but this varies.
CDC	The Connected Device Configuration is a framework for building J2ME applications.
Code Scattering	Having code pertaining to a concern distributed among several component.
Code Tangling	Having code for several different concerns gathered in a component.
Component Framework	A dedicated and focused architecture, usually around a few key mechanisms, and a fixed set of policies for mechanisms at the component level.
Concern	A concern or stakeholder concern is in the context of AOSD
CPL	Common Public License
Crosscutting Concern	A crosscutting concern is any concern or functionality that is secondary to the function of the business logic of the object and is present in multiple locations throughout the system. Examples of concerns that usually are crosscutting are: Authentication, logging, resource pooling, performance, storage management, data persistence, security, multithread safety, transaction integrity, error checking and most non-functional requirements.
GPL	GNU General Public License
GPRS	General Packet Radio Service is a mobile data service and a part of the GSM mobile standard.
Hook	Hooks are used in load-time and run-time weaving in AOP frameworks. A hook is the implementation of joinpoint. It is a call to an aspect manager which determines whether an aspect should be run at that point or not.
IDE	Integrated development Environment
Introduction	From AOSD: Introduction is a static addition to a class, interface or aspect that do not directly effect the object's behaviour. Adding a field or a method to a class is an example of an introduction. Introductions are sometimes referred to as intertype declarations.
J2ME	Java 2 Micro Edition

J2SE	Java 2 Standard Edition
JCP	Java Community Process
JIT Compiler	A Just-in-time compiler is a part of any Java Virtual Machine. It converts byte-code to the native machine code.
Joinpoint	From AOSD: A joinpoint is an identifiable point in the execution of a program. Examples of joinpoints are: method-calls, method executions, exceptions, reading or writing to a variable or constructor calls. Each framework support different types of pointcuts but the most common joinpoints that all AOP frameworks support are method calls. Methods abstract a functionality and
Joinpoint model	An AOP frameworks joinpoint model consists of all the joinpoints supported by that framework.
JSR	Java Service Request. The starting point of everything that eventually becomes a part of the standard Java Language.
JVM	Java Virtual Machine
JVMDI	Java Virtual Machine Debugger Interface
LGPL	GNU Lesser General Public License
MPL	Mozilla Public License
OO/OOD/OOSD	Object-Orientation, Object-Oriented design, Object-Oriented Software Development
Pointcut	A pointcut is a declarative statement which specifies one or more joinpoints. A pointcut can collect context, such as method arguments from the joinpoint. Pointcuts are sometimes referred to as pointcut designators.
UML	Universal Modelling Language
Weaving	Weaving is the process of integrating the aspect with the target object. Conversely unweaving is the process of separating the aspect from the target object. Weaving can be performed at compile-time, load-time or run-time. Compile-time weaving is referred to as static weaving, while load-time and run-time weaving are referred to as dynamic weaving.
WIFI/WLAN	Wireless Ethernet. IEEE 802.11b/g and other protocols used in wireless networks.
Wildcard	A wildcard is character that can be used to substitute for any other character or characters in a string.

Appendix C – Trace of Prototype 2

```
Context Change Detected:
security_requested changed from 0 to 100
Current Config: 0000 gives Utility: 83
tmp Config: 1000 gives Utility: 83
tmp Config: 0100 gives Utility: 83
tmp Config: 0010 gives Utility: 100
tmp Config: 0001 gives Utility: 66
Doing recursive call
tmp Config: 1010 gives Utility: 94
tmp Config: 0110 gives Utility: 94
tmp Config: 0010 gives Utility: 100
tmp Config: 0011 gives Utility: 77
Added Aspect SecurityAspect
The configuration is now: 0010 with Utility: 100
Context Change Detected:
power_available changed from 90 to 30
Current Config: 0010 gives Utility: 83
tmp Config: 1000 gives Utility: 66
tmp Config: 0100 gives Utility: 75
tmp Config: 0010 gives Utility: 83
tmp Config: 0001 gives Utility: 50
Doing recursive call
tmp Config: 1010 gives Utility: 77
tmp Config: 0110 gives Utility: 86
tmp Config: 0010 gives Utility: 83
tmp Config: 0011 gives Utility: 61
Doing recursive call
tmp Config: 1110 gives Utility: 83
tmp Config: 0110 gives Utility: 86
tmp Config: 0110 gives Utility: 86
tmp Config: 0111 gives Utility: 66
Added Aspect PowerConservationAspect
The configuration is now: 0110 with Utility: 86
Context Change Detected:
handsfree_requested changed from 0 to 100
Current Config: 0110 gives Utility: 69
tmp Config: 1000 gives Utility: 50
tmp Config: 0100 gives Utility: 58
tmp Config: 0010 gives Utility: 66
tmp Config: 0001 gives Utility: 66
Doing recursive call
tmp Config: 1010 gives Utility: 61
tmp Config: 0110 gives Utility: 69
tmp Config: 0010 gives Utility: 66
tmp Config: 0011 gives Utility: 77
Doing recursive call
tmp Config: 1011 gives Utility: 75
tmp Config: 0111 gives Utility: 83
tmp Config: 0011 gives Utility: 77
tmp Config: 0011 gives Utility: 77
Doing recursive call
tmp Config: 1111 gives Utility: 81
tmp Config: 0111 gives Utility: 83
tmp Config: 0111 gives Utility: 83
tmp Config: 0111 gives Utility: 83
Added Aspect HandsfreeAspect
The configuration is now: 0111 with Utility: 83
Context Change Detected:
security_requested changed from 100 to 0
Current Config: 0111 gives Utility: 83
tmp Config: 1000 gives Utility: 66
tmp Config: 0100 gives Utility: 75
tmp Config: 0010 gives Utility: 66
tmp Config: 0001 gives Utility: 83
Doing recursive call
tmp Config: 1001 gives Utility: 80
tmp Config: 0101 gives Utility: 88
tmp Config: 0011 gives Utility: 77
tmp Config: 0001 gives Utility: 83
Doing recursive call
tmp Config: 1101 gives Utility: 84
tmp Config: 0101 gives Utility: 88
tmp Config: 0111 gives Utility: 83
tmp Config: 0101 gives Utility: 88
Removed Aspect SecurityAspect
The configuration is now: 0101 with Utility: 88
Context Change Detected:
memory_available changed from 90 to 40
Current Config: 0101 gives Utility: 71
tmp Config: 1000 gives Utility: 66
tmp Config: 0100 gives Utility: 58
tmp Config: 0010 gives Utility: 50
tmp Config: 0001 gives Utility: 66
Doing recursive call
tmp Config: 1000 gives Utility: 66
tmp Config: 1100 gives Utility: 71
tmp Config: 1010 gives Utility: 61
tmp Config: 1001 gives Utility: 71
Doing recursive call
tmp Config: 1100 gives Utility: 71
tmp Config: 1100 gives Utility: 71
tmp Config: 1110 gives Utility: 58
tmp Config: 1101 gives Utility: 67
No improvement could be found. No reconfiguration needed.
The configuration is now: 0101 with Utility: 71
Context Change Detected:
power_available changed from 30 to 100
Current Config: 0101 gives Utility: 80
tmp Config: 1000 gives Utility: 83
tmp Config: 0100 gives Utility: 66
tmp Config: 0010 gives Utility: 66
tmp Config: 0001 gives Utility: 83
Doing recursive call
tmp Config: 1000 gives Utility: 83
tmp Config: 1100 gives Utility: 80
tmp Config: 1010 gives Utility: 77
tmp Config: 1001 gives Utility: 88
Doing recursive call
tmp Config: 1001 gives Utility: 88
tmp Config: 1101 gives Utility: 76
tmp Config: 1011 gives Utility: 75
tmp Config: 1001 gives Utility: 88
Added Aspect MemoryConservationAspect
Removed Aspect PowerConservationAspect
The configuration is now: 1001 with Utility: 88
```

Appendix D – Code of Prototype 2

```java
/*
 * Created on 11.jun.2005
 *
 *
 */
package diplom.prototype2.adaptationSystem;
import java.util.Hashtable;
/**
 * @author jc
 *
 *
 */
public class Component {

    private Hashtable properties;
    private Hashtable newProperties;

    public Component(Hashtable p){
        properties = p;
    }

    /**
     * @return Returns the properties.
     */
    public Hashtable getProperties() {
        return properties;
    }
}
```

```java
/*
 * Created on 11.jun.2005
 *
 *
 */
package diplom.prototype2.adaptationSystem;

import java.util.ArrayList;

/**
 * @author jc
 * @version 2
 *
 */
public class AspectAdaptationManager implements ContextChangeListener,
SimulationManager{
    private Hashtable _properties;
    private Hashtable _contextStorage;
    private String _contextFile = "context.trix";
    private Context _context;
    private String _logFile = "logfile.trix";
    private SimulatorLog _log;
    private Hashtable _contextTypeTable = new Hashtable();
    private AspectLibrary _al;
    private ArrayList _propertyNames;
    private Evaluator _evaluator;
    private Config _currentConfig;
    private ArrayList _componentList;
    private MeasurementSystem app;

    public AspectAdaptationManager(AspectLibrary aspectLibrary){
        _componentList = new ArrayList();
        _propertyNames = new ArrayList();
        _al = aspectLibrary;
        _evaluator = new Evaluator(this);


        //initializing and populating properties about the application
        _properties = new Hashtable();
        _contextStorage = new Hashtable();
        _propertyNames.add("memory_required");
        _propertyNames.add("bandwidth_required");
        _propertyNames.add("handsfree_offered");
        _propertyNames.add("power_required");
        _propertyNames.add("latency");
        _propertyNames.add("security");

        //collected from TestLocalOnlyWithSimulator.java
        XMLContextParser parser;

        try{

            // Initialisation
            // Create the context description parser
            //  The file to be parsed was given as argument
            parser = new XMLContextParser (_contextFile);
            _context = parser.getContext ();

            // Now, transfrom this context to the simplified platform context
representation
            setPlatformContext(_context); //tror ikke den her trengs. joda


            // Create log and save initial context
            //  The log file was given as argument
            _log = new SimulatorLog (_logFile, _context);
            // Inilialize Context log listener
            _log.listenContextChange();

            //setupAndStartDemoApp();
            //  Create gui
            new SimulatorGui(this, _context);
            // Terminate

        } catch (RuntimeException r) {
            System.exit (1);
        }
    }


    public void contextChange(ContextEvent e){
        //System.out.println(e.getPropValue() +"\n" + e.getAttName());
        System.out.println("Context Change Detected:");
        int valOld = ((Integer)_contextStorage.remove(e.getAttName())).intValue();
        int valNew = Integer.parseInt(e.getPropValue());
        System.out.println(e.getAttName() + " changed from " + valOld + " to " +
 valNew);
        _contextStorage.put(e.getAttName(), new Integer(valNew));
        //System.out.println("------------Context Updated------------");
        //currentConfig = evaluator.evaluate(currentConfig, _contextStorage);//evaluate
current config
        _currentConfig = _evaluator.evaluate(_currentConfig,
 _contextStorage);//evaluate current config
        System.out.print("Current Config: ");
        for(int i=0; i<_currentConfig.getConfig().length;i++){
            System.out.print(_currentConfig.getConfig()[i]);
        }
        System.out.println(" gives Utility: " + _currentConfig.getUtility());

        int[] zeros = new int[_al.getAspectLib().size()];
        for (int i=0; i<_al.getAspectLib().size();i++){zeros[i] = 0;}//create an array
of zeros

        //Config newConfig = adaptGreed(new
Config((int[])_currentConfig.getConfig().clone()));//change to adaptAll() to use the
complete adaptation algorithm
        Config newConfig = adaptGreed(new Config(zeros));//change to adaptAll() to use
the complete adaptation algorithm
        if (_currentConfig.getUtility()<newConfig.getUtility()){//if a better
configuration has been found
            reconfigure(newConfig);//use this configuration
        }
        else System.out.println("No improvement could be found. No reconfiguration
needed.");
        System.out.print("The configuration is now: ");
        for(int i=0; i<_currentConfig.getConfig().length;i++){
            System.out.print(_currentConfig.getConfig()[i]);
        }
        System.out.println(" with Utility: " + _currentConfig.getUtility());
    }

    public Config adaptGreed(Config c){
        c = _evaluator.evaluate(c, _contextStorage);//evaluate input config

        int[] zeros = new int[_al.getAspectLib().size()];
        for (int i=0; i<_al.getAspectLib().size();i++){zeros[i] = 0;}//create an array
of zeros
        Config baseConfig = new Config(zeros);//create a config with no aspects
installed
        baseConfig = _evaluator.evaluate(baseConfig, _contextStorage);//evaluate base
config
```

```
        int best = c.getUtility();//initializing best-utility variable
        int newAspect = -1;
        for (int i=0; i<_al.getAspectLib().size();i++){//for all aspects
            int[] array = (int[]) c.getConfig().clone();//create an array of the
current config
            array[i]=1;//add aspect i
            Config tmp = new Config(array);//create a config object
            tmp = _evaluator.evaluate(tmp, _contextStorage);//evaluate current config +
aspect i

            System.out.print("tmp Config: ");
            for(int k=0; k<tmp.getConfig().length;k++){
                System.out.print(tmp.getConfig()[k]);
            }
            System.out.println(" gives Utility: " + tmp.getUtility());
            if (tmp.getUtility()>best){
                best = tmp.getUtility();
                newAspect = i;
            }
        }
        if (newAspect!=-1){//if found an improvement
            c.getConfig()[newAspect] = 1;//current config +new Aspect
            c = _evaluator.evaluate(c, _contextStorage);
            System.out.println("Doing recursive call");
            return adaptGreed(c);
        }
        return c;


    }



    public void reconfigure(Config newConfig){
        for (int i=0; i<newConfig.getConfig().length;i++){
            if (newConfig.getConfig()[i]==_currentConfig.getConfig()[i]){//No change to
aspect
                //do nothing
            }
            else if(newConfig.getConfig()[i]==1 && _currentConfig.getConfig()[i]
==0){//Aspect should be added
                weaveInAspect((AspectProperties)_al.getAspectLib().get(i));

            }
            else if(newConfig.getConfig()[i]==0 && _currentConfig.getConfig()[i]
==1){//Aspect should be removed
                weaveOutAspect((AspectProperties)_al.getAspectLib().get(i));

            }
        }
        _currentConfig = newConfig;
    }
    public void weaveInAspect(AspectProperties aspect){
        System.out.println("Added Aspect " + aspect.getName());
        //the code for adding an aspect
        //the code for weaving in changes to the model
    }
    public void weaveOutAspect(AspectProperties aspect){
        System.out.println("Removed Aspect " + aspect.getName());
        //the code for removing an aspect
        //the code for weaving in changes to the model
    }

    public void registerComponent(Component c){
        _componentList.add(c);
    }
    public void unRegisterComponent(Component c){
        _componentList.remove(c);
    }
    public Hashtable sum(Hashtable h1, Hashtable h2){
        Hashtable h3 = new Hashtable();
        for (int i=0; i<_propertyNames.size(); i++){//for each property

            Integer val1 = (Integer) h1.get(_propertyNames.get(i));
            Integer val2 = (Integer) h2.get(_propertyNames.get(i));
            Integer val3 = new Integer(val1.intValue() + val2.intValue());
            h3.put(_propertyNames.get(i), val3);
        }
        return h3;
    }

    public void setPlatformContext(Context context) {
        // Convert to platform hashtable
        //Hashtable tmp = new Hashtable();
        for (int i = 0; i < context.contextElements.size(); i++) {
            ContextElement el = (ContextElement) context.contextElements.elementAt(i);
            el.addContextChangeListener(this);

            for (int j = 0; j < el.contextAttributes.size(); j++) {
                ContextAttribute at = (ContextAttribute)
el.contextAttributes.elementAt(j);
                String name = at.getName();
                String atType = at.getType();
                String value = el.getContextAttributeProperty(name, "current");
                Object transValue = convertToType(value, atType);
                if (transValue != null) {
                    //tmp.put(name, transValue);
                    _contextStorage.put(name, transValue);
                }
            }
        }
    }
    public void exit() {
        _log.terminateLogging();

        System.exit(0);
    }
    protected Object convertToType(String value, String atType) {
        Object transValue = null;
        if (atType.equalsIgnoreCase("integer")) {
            try {
                transValue = new Integer(value);
            }
            catch (Exception ex) {};
        }
        else if (atType.equalsIgnoreCase("enumerated")) {
            transValue = value;
        }
        else if (atType.equalsIgnoreCase("string")) {
            transValue = value;
        }
        else if (atType.equalsIgnoreCase("boolean")) {
            transValue = Boolean.valueOf(value);
        }
        return transValue;
    }
    /**
     * @return Returns the _componentList.
     */
    public ArrayList get_componentList() {
        return _componentList;
    }
```

```java
    /**
     * @return Returns the _propertyNames.
     */
    public ArrayList getPropertyNames() {
        return _propertyNames;
    }
    /**
     * @return Returns the al.
     */
    public AspectLibrary getAl() {
        return _al;
    }

    public void initialize(){
        app = new MeasurementSystem();
        _componentList.add(app);
        _componentList.add(app.getCom());
        _componentList.add(app.getDriverInstaller());
        _componentList.add(app.getGui());
        _componentList.add(app.getSensorDetector());
        AspectProperties mca = new MemoryConservationAspect(app.getSensorDetector(),
 app);
        AspectProperties pca = new PowerConservationAspect(app.getCom());
        AspectProperties sa = new SecurityAspect(app.getCom());
        AspectProperties ha = new HandsfreeAspect(app.getGui());
        _al.registerAspect(mca);
        _al.registerAspect(pca);
        _al.registerAspect(sa);
        _al.registerAspect(ha);
        int[] zeros = new int[_al.getAspectLib().size()];
        for (int i=0; i<_al.getAspectLib().size();i++){zeros[i] = 0;}//create an array
of zeros
        _currentConfig = new Config(zeros);//create a config with no aspects installed
    }

    public static void main(String args[]){
        AspectLibrary al = new AspectLibrary();
        AspectAdaptationManager trixy = new AspectAdaptationManager(al);
        trixy.initialize();

    }
}
```

```java
/*
 * Created on 19.jun.2005
 *
 *
 */
package diplom.prototype2.adaptationSystem;
import java.util.Hashtable;
/**
 * @author jc
 *
 * Effect is a dataobject that contains a pointer to the component it affects and the
 value of the effects.
 * One effect-object exist for each component affected.
 */
public class Effect {
    private Hashtable effects;
    private Component componentAffected;

    public Effect(Component cA, Hashtable e){
        effects = e;
        componentAffected = cA;
    }

    /**
     * @return Returns the componentAffected.
     */
    public Component getComponentAffected() {
        return componentAffected;
    }
    /**
     * @return Returns the effects.
     */
    public Hashtable getEffects() {
        return effects;
    }
}
```

```java
/*
 * Created on 15.jun.2005
 *
 *
 */
package diplom.prototype2.adaptationSystem;

/**
 * @author jc
 *
 * The config is a data object that contains a configuration, represented
 * as a one-dimensional matrix and the corresponding utility of
 * that configuration in a given context
 */
public class Config {
    int utility;
    int[] config;

    public Config(){

    }
    public Config(int ut, int[] cfg){
        utility = ut;
        config = cfg;
    }
    public Config(int[] cfg){
        config = cfg;
    }
    /**
     * @return Returns the config.
     */
    public int[] getConfig() {
        return config;
    }
    /**
     * @return Returns the utility.
     */
    public int getUtility() {
        return utility;
    }
    /**
     * @param config The config to set.
     */
    public void setConfig(int[] config) {
        this.config = config;
    }
    /**
     * @param utility The utility to set.
     */
    public void setUtility(int utility) {
        this.utility = utility;
    }

}
```

```
/*
 * Created on 11.jun.2005
 *
 *
 */
package diplom.prototype2.adaptationSystem;

import java.util.Hashtable;
/**
 * @author jc
 * @version 2
 *
 */
public class Evaluator {
    AspectAdaptationManager manager;
    public Evaluator(AspectAdaptationManager aam){
        manager = aam;
    }

    public Config evaluate(Config cfg, Hashtable contextStorage){
        int[] propertySums = new int[manager.getPropertyNames().size()];//create
temporary variable
        Hashtable summedProperties = new Hashtable();
        for(int j=0; j<manager.getPropertyNames().size();j++){//for all properties
            String pro = (String) manager.getPropertyNames().get(j);
            for (int i=0; i<manager.get_componentList().size();i++){//for all
components
                Component c = (Component) manager.get_componentList().get(i);
                Integer integer = (Integer) c.getProperties().get(pro);
                propertySums[j] += integer.intValue();
                summedProperties.put(manager.getPropertyNames().get(j), new
 Integer(propertySums[j]));


            }
        }


        for(int i=0; i<manager.getAl().getAspectLib().size();i++){//get effects of
aspects
            if(cfg.getConfig()[i]==1){
                ArrayList effects =
 ((AspectProperties)manager.getAl().getAspectLib().get(i)).getEffects();
                for(int j=0; j<effects.size();j++){
                    summedProperties =
 manager.sum(summedProperties,((Effect)effects.get(j)).getEffects());
                }

            }
        }
        int utilities[] = new int[manager.getPropertyNames().size()];//create temporary
variable for utility for each property
        utilities[0] = calculateMemoryUtility(summedProperties,
 contextStorage);//calculate partial utilities
        utilities[1] = calculateBandwidthUtility(summedProperties, contextStorage);
        utilities[2] = calculateHandsfreeUtility(summedProperties, contextStorage);
        utilities[3] = calculatePowerUtility(summedProperties, contextStorage);
        utilities[4] = calculateLatencyUtility(summedProperties, contextStorage);
        utilities[5] = calculateSecurityUtility(summedProperties, contextStorage);
        int utility = 0;
        for (int i=0;i<6;i++){
            utility += utilities[i];
        }
        utility = utility/6;
        cfg.setUtility(utility);
        return cfg;
```

```java
    }
    public int calculateMemoryUtility(Hashtable s, Hashtable cS){
        int available = ((Integer)cS.get("memory_available")).intValue();
        int required = ((Integer)s.get("memory_required")).intValue();
        if ((available-required)>10){
            return 100;
        }
        else if((available-required)>0){
            return (10*(available-required));
        }
        else return  0;
    }
    public int calculateBandwidthUtility(Hashtable s, Hashtable cS){
        int available = ((Integer)cS.get("network_bandwith")).intValue();
        int needed = ((Integer)s.get("bandwidth_required")).intValue();

        if(available>=needed){
            return 100;
        }
        else if(available==0){
            return 0;
        }
        else return (int)(100*(double)available/needed);
    }
    public int calculateHandsfreeUtility(Hashtable s, Hashtable cS){
        int handsfree_requested = ((Integer)cS.get("handsfree_requested")).intValue();
        int handsfree_offered = ((Integer)s.get("handsfree_offered")).intValue();

        if(handsfree_requested == handsfree_offered){
            return 100;
        }
        else {
            return 0;
        }

    }
    public int calculatePowerUtility(Hashtable s, Hashtable cS){
        int powerState = ((Integer)cS.get("power_available")).intValue();
        int powerUse = ((Integer)s.get("power_required")).intValue();
        if (powerState-powerUse>20){
            return 100;
        }
        else if(powerState<=powerUse){
            return  0;
        }
        else return  5*(powerState-powerUse);
    }
    public int calculateLatencyUtility(Hashtable s, Hashtable cS){
        int tolerated = ((Integer)cS.get("latency_tolerated")).intValue();
        int latency = ((Integer)s.get("latency")).intValue();

        if(latency<=tolerated){
            return 100;
        }
        else{
            return (tolerated*100)/(latency);
        }
    }

    public int calculateSecurityUtility(Hashtable s, Hashtable cS){
        int security_requested = ((Integer)cS.get("security_requested")).intValue();
        int security = ((Integer)s.get("security")).intValue();

        if(security>=security_requested){
            return 100;
        }
```

```
          else{
              return 0;
          }
      }

  }
```

```java
package diplom.prototype2.adaptationSystem;
import java.util.ArrayList;
/**
 * @author jc
 *
 * The aspectLibrary keeps a registry of the available aspects and their
 * metainformation.
 *
 */
public class AspectLibrary {
    private ArrayList aspectLib;
    private ArrayList active;
    public AspectLibrary(){
        aspectLib = new ArrayList();
        active = new ArrayList();
    }
    public void registerAspect(AspectProperties aspect){
        aspectLib.add(aspect);
    }

    /**
     * @return Returns the aspectLib.
     */
    public ArrayList getAspectLib() {
        return aspectLib;
    }
    public void weaveAspect(AspectProperties aspect){
        System.out.println("Weaving aspect " + aspect.getName());
        active.add(aspect);
    }
    public void unWeaveAspect(AspectProperties aspect){
        System.out.println("Unweaving aspect " + aspect.getName());
        active.remove(aspect);
    }

}
```

```java
/*
 * Created on 15.mai.2005
 *
 *
 */
package diplom.prototype2.adaptationSystem;
import java.util.ArrayList;
/**
 * @author jc
 *
 * AspectProperties must be implemented by aspects to enable the aspectLibrary to acces
 the
 * metainformation about the aspects.
 */
public interface AspectProperties {



    public String getName();
    public ArrayList getEffects();

}
```