Modeling for Pervasive Computing: Organizational aspects of system analysis and design

Mads Ingstrup
ingstrup@mip.sdu.dk

Masters Thesis
Supervisor: Palle Nowack

THE MAERSK MC-KINNEY MOLLER INSTITUTE
FOR PRODUCTION TECHNOLOGY
UNIVERSITY OF SOUTHERN DENMARK, ODENSE

June 1, 2003
Contents

Preface vi
Organization of this document vii

INTRODUCTION, PROBLEM STATEMENT AND RESULTS, 1

A Motivation: Modeling for Pervasive Computing 1
B Analysis: Conceptual foundation of the project 2
   B.i Systems 3
   B.ii Problem and Application Domains 3
   B.iii Interface, Model and Functionality 4
   B.iv Software Development Process 5
   B.v Abstraction 5
   B.vi Design Philosophy 6
   B.vii Pervasive Computing 7
   B.viii Convergence of pervasive computing 9
C Delimitation and Formal Goals 10
   C.i Modeling and Methodology for pervasive computing 10
   C.ii The influencing forces on software 10
   C.iii Formal Goals 11
D Approach 12
E Results 13
   E.i The arguments for the hypotheses 14
F Conclusion 20
   F.i The Role of Pervasive Computing 20
   F.ii Experimental and Analytical methods of work 20
   F.iii Future Work 21

CHAPTER 1. Applying the Unified Process, 23

1.1 Purpose 23
1.2 Method 23
   1.2.1 Focus on Business, Use Case and Analysis Models 24
1.3 Three classes of models 25
  1.3.1 Business Model 25
  1.3.2 Requirements Model 26
  1.3.3 Analysis Model 26
1.4 BM, RM and AM for the Mitas 27
  1.4.1 Business Model 27
  1.4.2 Requirements/Use Case Model 29
  1.4.3 Analysis Model 30
  1.4.4 Design Model, partly 30
1.5 Observations made about UP and development 31
  1.5.1 Observations pertaining to BM and RM 32
  1.5.2 Observations pertaining to AM 32
1.6 Conclusion 33

CHAPTER 2. Deriving criteria for a modeling technique, 35

2.1 Purpose 35
2.2 Method 35
  2.2.1 Consequences of extension instead of a new system 36
2.3 The development case 36
2.4 Observations, Analysis and Criteria 38
  2.4.1 Concrete observations about UML and UP 38
  2.4.2 Observations about the general nature of the system 42
  2.4.3 Analysis of observations and formulation of criteria 43
2.5 Conclusion 44

CHAPTER 3. Organizational structure and software qualities, 47

3.1 Purpose 47
3.2 Method 47
3.3 Comparison between software systems and organizations 49
3.4 Existing work on system vs organization 50
  3.4.1 Global Analysis 50
3.5 Analysis of each stereotypical configuration 51
  3.5.1 The Simple Structure 52
  3.5.2 The Machine Bureaucracy 52
  3.5.3 The Professional Bureaucracy 54
  3.5.4 The Divisionalized Form 56
  3.5.5 The Adhocracy 57
3.6 Conclusion 58
  3.6.1 Discussion 59
CHAPTER 4.  A modeling technique for pervasive systems, 61

4.1 Method 61
4.2 The Philosophy of modelling 62
   4.2.1 What is a model? 62
   4.2.2 Models of reality 63
   4.2.3 Models of software 65
4.3 Modeling Primitives 66
   4.3.1 Tangible Object 66
   4.3.2 Physical Surroundings 68
   4.3.3 Break-down centered business model 72
4.4 Conclusion 75

CHAPTER 5.  The Interaction Widget Pattern, 77

5.1 Problem 77
5.2 Solution 78
5.3 Consequences 80
5.4 Known Uses 82
5.5 Related Patterns 82

APPENDIX A.  Reflections on the observations of activity one, 83

A.1 Micro iterations 83
   A.1.1 Reflections on method of work 83
A.2 Reflection on results 84
   A.2.1 Regarding shortcomings detected during analysis 84
   A.2.2 Primary and Secondary Use cases 85
   A.2.3 Comparison of properties of BM, RM, AM (DM) 86

APPENDIX B.  Activities and Calendar, 87

B.1 Project Calendar 90

APPENDIX C.  Properties of pervasive computing, 91
BIBLIOGRAPHY, 93

INDEX, 99

APPENDIX D. Development Report for activity I, 101

APPENDIX E. Development Report for activity II, 101+33
Preface

This document is written in partial fulfilment for the degree cand.polyt, together with an oral presentation and exam.

It has been written alongside performing the work it describes, in the period from September 1st 2002 to June 1st 2003.

I would like to thank my supervisor Palle Nowack for helping me shape and articulate the activities and thoughts presented herein. Also, I would like to thank Daniel May for, in the role of shepherd, helping me improve the description of the pattern presented in chapter 5.

Finally, I would like to thank for the Oticon scholarship that was granted to me in support of this project.

MADS INGSTRUP
MAY 2003
Contents

Organization of this document

This section describes the purpose and content of each of the following chapters.

Introduction, Problem Statement and Results gives an overview of the project. It does so by describing the overall goal and the five activities undertaken in pursuit of this goal. The concepts central to the project are also discussed. Next, the chapter contains a set of theses that have been formulated based on the insights gained during the activities. The chapter closes with the conclusion of the project.

Chapter 1 describes and presents the results of the first of the five activities. In doing so the application domain considered in the concrete activities of this project is introduced.

Chapter 2 presents the attempt to use Unified Process to develop a pervasive system, and the observations made during this work. It also formulates some criteria for a modeling technique for pervasive systems.

Chapter 3 contains a comparison of software systems and organizations, and it contains an analysis of each of Mintzberg’s five stereotypical organizations. Based on this it is suggested for each stereotype what qualities should be possessed by a software system to be used in organizations of this stereotype.

Chapter 4 starts with a presentation of a philosophical view of models. It then proceeds with applying this to form a modeling technique for pervasive systems, which adhere to the criteria in chapter 2.

Chapter 5 presents a pattern for designing user interfaces. It is general enough to be usable in the creation of pervasive systems. The pattern has been accepted for publication as: Interaction Widgets. Ingstrup, Mads. Proceedings of the 8th European Conference on Pattern Languages of Programs, June 25th – 29th 2003, Irsee, Germany.

Appendix A contains some observations and reflections made during activity one. They are included because they are a byproduct of the activity, but placed in an appendix because they are secondary to the goal of the project.

Appendix B contains the timeplan for the project.

Appendix C contains a table that explains the properties of pervasive computing with those of traditional systems. The table is copied from [40].

Appendix D is the complete development report for the first activity; parts of it appear in chapter 1.

Appendix E is the development report for the second activity, as performed in activity 2. It is described relative to the first development report in appendix C. The activity as a whole is presented in chapter 2.
Introduction, Problem Statement and Results

This chapter contains a presentation of the subjects, approach and results of this project as well as the conclusion. The first section, A, is a description and motivation of the hypotheses that initiated the project. In section B the initiating hypotheses are analyzed and the conceptual foundation of the project is presented by defining and discussing central concepts. Section C states the formal goals of this project based on the concepts in the preceding section. Next, section D describes the approach that has been taken to reach these goals. In section E the results of this project are presented in the form of a number of hypotheses which are argued by referring to chapters 1–5 and appendices. Finally, section F concludes this project and discuss some possibilities for future work in the subjects dealt with.

A Motivation: Modeling for Pervasive Computing

This is a project about software for pervasive computing. Pervasive or Ubiquitous computing was stated more than a decade ago by Mark Weiser [70] as the vision of computers embedded in everything, becoming an increasingly tacit and natural part of our lives, woven “into the fabric of everyday life until they disappear.”[70].

When much of the hardware needed to build pervasive systems exists and the vision has yet to be realized[57] it seems that developing the right software is the bottleneck in achieving this realization. This is unfortunate since software has traditionally been the driving force in introducing and spreading information technology. After all, as noted by [71] applications are the whole point of ubiquitous computing. An example is that for most PC users the incentive to buy a new piece of hardware is the possibilities provided by the latest applications, e.g. a 3D game[20] etc.

Pervasive computing systems are expectedly complex [7, 22, 57]. Small and simple programs can be written without the support of a formal process and modeling, but the more complex the software the greater the need for an organized approach; a development method build around a purposeful set of models expressed in an understandable format. Pervasive systems are expected to exhibit intelligent, anticipating and adaptive behaviour [40] and increasingly pervade and be in-
tegrated with our physical surroundings, while at the same time remaining largely tacit. One of the views that initiated this project was the intuitive feeling that these properties of pervasive computing meant that

- the analysis and design methods and modeling tools used for traditional administrative software systems are insufficient for building pervasive systems.

One example of this insufficiency which was found in the project, was a lack of means to express and reason about the physical and locational aspects of a pervasive system.

That computing is ubiquitous implies that it has a greater presence in our lives; the activities we engage in and the things surrounding us are pervaded by this form of computing. While this opens up many possibilities it also enable these systems to be more obstructing and in the way. Certainly a goal must be not to build a pervasive system that has “to be worked around” for people to obtain what they want from it. Yet, for traditional administrative systems this is not unheard of [60, 18, 61].

Studies such as [45, 44] exemplifies a failure to address the “soft” issues in the organization of the users, thus not rewarding “the need to align the technical and the social”[45]. This leads to the second hypothesis that initiated this project, namely that

- for a software system and in particular a pervasive one, the system’s alignment with the organizational and social situation in its context is crucial.

The following sections will analyze these initiating hypotheses by presenting the concepts central to their investigation as it has been undertaken in this project.

B Analysis: Conceptual foundation of the project

This section introduces the central concepts of the project and their relevance to the initiating hypotheses. The words emphasized in the following paragraph are those defined and discussed in the subsections B.i–B.viii.

Both of the initiating hypotheses are about analysis and design of software for pervasive computing. Software is treated using the notions of a system and its environment, these are what is being analyzed and designed. The problem and application domains are abstractions of the environment, and used to analyze and model it. It is through these abstractions that the organization of the users are taken into account in a development process, to the degree this happens. In designing and modeling a system it can be usefully seen as having the three parts model, interface and functionality. The processes and models are abstractions which reflect the kinds of systems they are intended for, e.g. administrative systems or realtime systems—or as per the initiating hypotheses above, pervasive computing systems. Depending on what qualities are desired in the system being build, different design philosophies may be chosen. The convergence of pervasive computing towards
Weiser’s vision implies that the methodology and models for creating pervasive systems are likely evolve alongside this convergence.

**B.i Systems**

When the initiating hypotheses mention analysis and design, what is being analyzed and designed is a software system and its environment. The following definitions of a system and its environment are both taken from [3, p. 4].

**DEFINITION 1.1**

A system is defined as a coherent set of interdependent components that exists for some purpose, has some stability, and can be usefully viewed as a whole.

**DEFINITION 1.2**

The environment of a system is anything outside the system that has an effect on the way the system operates.

For software it makes sense to broaden the definition of its environment to include anything outside the system that the system has an effect on.

In the development process the system’s environment is considered through the abstractions of problem and application domains, which are explained next.

**B.ii Problem and Application Domains**

Part of the models referred to in the first initiating hypothesis are models of two abstractions, the problem domain and the application domain. They are considered separate because they capture two complementary views that are used for different purposes during analysis and design. The concepts are defined by [38, p. 6] (translated from Danish) as:

**DEFINITION 1.3**

The Problem Domain is the part of the software’s environment that is administered, monitored, or controlled using a system.

**DEFINITION 1.4**

The Application Domain is the organization that administers, monitors, or controls a problem domain.

The domains and their relations are illustrated in figure i.1. The analysis and models of these domains form the basis for the design of the system, but this does little to take into account the social and organizational issues in the problem and application domains. The design of the system is structured relative to the concepts of interface, model, and functionality.
B.iii Interface, Model and Functionality

Many of the models and methods (UP, OOA&D) referred to in the initiating hypothesis assume a view of a system as having three parts, and this assumption is manifested in the analysis and design workflows during which they are created. The three parts are the Interface, Model and Functionality (or Control). In figure i.1 that illustrates a system’s relation to its problem and application domains, the system is shown as conforming with this general architecture as proposed in [38, p. 10]. The Unified Process [27] also make the distinction between these parts of a system, but on the level of classes. This means that while the interface, model and functionality aspects of a system are still separate concerns, the actual components of implementation may each contain both model, interface and functionality, c.f. the PAC architectural pattern [8]. The three components are defined as (again translated from Danish).

DEFINITION 1.5

Model component: The part of a system which realizes a model of a problem domain. [38, p. 230]

The analysis of the problem domain results in a domain model. This forms the basis for the model part of the system which is thus derived from the problem domain as illustrated in figure i.1.

DEFINITION 1.6

Function component: The part of a system which realizes the functional requirements. [38, p. 246]

The functional requirements are often captured using use cases; a use case model is a model of the application domain.

DEFINITION 1.7

Interface component: The part of a system which makes the model and functionality of a system accessible to actors. [38, p. 147]
The organization of the interface component reflects the set of distinct roles specified by actors. The set of actors is an abstraction of the set of users. The interactions between the users (actors) and the system is specified by the use case model.

In this project about models used to build pervasive systems, we want to characterize the observed differences in modeling traditional and pervasive systems. The distinction between these three aspects or parts of a system makes this characterization easier, as it can be done relative to distribution and relations among them.

**B.iv Software Development Process**

The concept of a development process is central to this project because it is difficult to separate models from modeling—how the models are created. The concept is also necessary because it is where to specify how to treat the effects from a system’s environment on the system. Thus the process is the area of concern if a better alignment between a system and the organization of its users is required, the importance of which was stated in the second hypothesis. Jacobson et al. gives the following definition:

> A process defines *who* is doing *what when* and *how* to reach a certain goal.

In software engineering the goal is to build a software product or to enhance an existing one.

... 

A software development process should also be capable of evolving over many years. During this evolution it should limit its reach at any given point in time to the realities that technologies, tools, people, and organizational patterns permit. ([27] p. xviii)

Using a stable development process will help ensure that a project runs somewhat smoothly. A development process such as the Unified Process, which will be explained later in this report, must possess some degree of general applicability. One may even take the view that it is a generic project; an abstraction over a set of concrete projects; in a sense a framework for projects. Yet a development process is both concrete and general at the same time. It is general in that it does not place many restrictions on the kind of systems or structures it can be used to develop. But it is concrete in defining what must be done to obtain the models that are the results of each workflow. This is achieved by discussing the models on a meta-level, referring to classes, packages, use cases and other concepts rather than their instances.

In the view of a process as an abstraction over projects, the assumed meaning of the word *abstraction* is as discussed next.

**B.v Abstraction**

Abstraction is both a verb and a noun. A process may be seen as an abstraction, but creating models during analysis and design requires *abstraction* to deal with
the complexity resulting from the large amount of details in the problem and application domains. Budd [6] defines the act of abstraction as:

**Definition 1.8**

**Abstraction** is the purposeful suppression, or hiding, of some details of a process or artifact, in order to bring out more clearly other aspects, details, or structures. [6, p. 26]

So to abstract is the act of creating an abstraction. From a mathematical point of view we may see the process or artifact as a set of details, and the abstraction as a subset of this set. But this view implicitly assumes that a process or artifact are discrete entities; this assumption is suitable for abstraction of programmatic entities, but perhaps not for a real world process. The above definition is not clear at that point, but concerning the usage of the concept in this writing it is concrete enough.

With this meaning of an abstraction, the concept of a design philosophy can be characterized by its relation to a process.

**B.vi Design Philosophy**

Suppose a development process is viewed as an abstraction over projects. In any abstraction something is in a sense left out compared to what it is an abstraction of. For a process it would be that which distinguishes the projects; and that which distinguishes a project from a process. If “something” is left out, one may ask what this something is and where it comes from when a concrete project is carried out using a process. It is believed that part of the answer is what will be called a design philosophy. So part of a design philosophy is a way of doing something, perhaps specified as a set of formal activities or only more generally as a state of mind to be in or a perspective to employ in the design choices made.

Mathiassen et al. [38] gives two examples of philosophies that constitute the extremes of a scale. The first philosophy is to make a system as automated as possible, essentially reducing the users to someone who provides information whenever prompted. The second philosophy is to make an application that is like a toolbox, allowing a user great discretion in carrying out the tasks supported by the application. So another part of a design philosophy is some goals about what qualities the final product should have.

A design philosophy may be evident from the formalized requirements, but not necessarily. Consider for instance the above two philosophies, and suppose a user requires support for some task. This user may be able to describe rigorously how the task is carried out, thus enabling a highly automated system by philosophy one. But if the way this task is performed changes sometimes, even if rarely, then philosophy two might be better. So while the choice of philosophy should be grounded in analysis of the software’s context the development process may not address how to make this choice.
As an example of a design philosophy consider radical tailorability described by Malone [37], where the goal is almost extreme flexibility in letting the users tailor the application, somewhat aligned with philosophy two from the previous paragraph. Bieber [4] advocates hypermedia, a design philosophy by which users should be able to navigate the intuitive relationships that exist between the different kinds of useful information held by an application. Hallnäs and Redström [20] describes slow technology as a design philosophy for pervasive systems, where the goal is for the final system to inspire reflection. Design philosophies thus seem to be shaped around goals of achieving a particular quality in the final product. The concept is not as mature as other concepts used here, but the following definition will be assumed:

DEFINITION 1.9
A design philosophy consists of a goal for some qualities to be present in the final product being developed, and a specified way of achieving this goal (e.g. particular design patterns, a general perspective to employ in making design choices, some specific activities).

Given that design philosophies are in a way orthogonal to the development process used, it seems that they provide a way of forming the system that can be chosen depending on the organization it should be used in.

B.vii Pervasive Computing

Both initiating hypotheses are stated based on the nature of pervasive computing.

The particular term pervasive computing seems to have been used first by IBM, see [74]. But the idea was first described by Mark Weiser in 1991, [70]. He introduced ubiquitous computing as the vision of computers melting into our world, the use of them becoming an increasingly natural and tacit part of our lives; computers should be woven “into the fabric of everyday life until they disappear.”[70]. Other descriptions make use of metaphors to convey the idea, e.g. “people embedded in a sea of computation” [42]. Aligned with these ideas is the name ambient computing [10].

Although pervasive and ubiquitous computing are often taken to mean the same, this is not always the case. [36] articulates a distinction between pervasive, ubiquitous, mobile and traditional computing systems using a figure like i.2. Here, pervasive computing does not imply mobility, but only embeddedness. According to the figure, pervasive systems are as static as traditional systems. The mobility is an important aspect, since adaptability to context is only necessary if the context changes. But purely mobile systems (e.g. current cell phones, PDAs) are not sensitive to their context; they are mobile but not embedded in the environment. To match Weiser’s vision, the system must have qualities of both mobility and embeddedness. In relation figure i.2, this project is about ubiquitous computing.

What becomes clear by studying the works of others in this area is that the technology is still at its dawn, but also that it is very much in growth as a research
area. Unfortunately it seems that there are no commonly accepted definition of what it is. Recognizing this, the two terms pervasive and ubiquitous computing will be used interchangeably in this project. What comes closest to an agreed upon description is the one above, presented in Weiser’s original 1991 article, since this is referred to in a very high percentage of articles about pervasive computing.

A study such as [47] also indicates that there is much more to building pervasive systems than the challenges being addressed in today’s development methods. This is to some degree already being recognized to be the case for more traditional systems [44, 45].

In the remainder of this writing, the nature of pervasive computing will play an important role, since this is exactly what the project aims at understanding. Kristensen et al. [40] has proposed a set of properties of pervasive systems. These are helpful in describing more concretely the respects in which pervasive and traditional systems are different. The properties are Intention, History, Dynamic, Personalized, Adaption, Anticipation and Time & Space. The explanations of the properties are replicated in the table in appendix [?], as appearing in [40].

A brief point about the dynamic property should be made here, since it is needed later. The system should also be accessible through interaction mechanisms not originally deployed with the system, or present at the time of its deployment. By interaction is meant that specified by a use case. Though access through new interaction mechanisms can be considered new interactions, it is strictly only a requirement that known interactions can be accessed this way. As such this requirement is weaker than the one foregrounded in the table.

These properties are descriptive of what pervasive computing is envisioned to be, but give little guidance for how to make a system that exhibits them. They seem to more or less specify that pervasive systems should be intelligent. It is clear that a pervasive system in a sense is more than the sum of its parts [57], but actual intel-
ligence in the human sense is perhaps not necessary—e.g. a pervasive system does not need conscience. Instead we may observe that some of the qualities that can be achieved in today's software would thirty years ago have been hypothesized to require intelligence—e.g. be able to speak to a computer or the behaviour exercised by non-player-characters in computer games.

The properties in the table in appendix C does not seem to address the way that pervasive computing has an increased presence in our surroundings, embedded into everything. A traditional workstation also has a physical presence, a screen, mouse, speakers, keyboard etc. Pervasive computing is different because its tangible objects have a purpose in themselves, whereas a screen, keyboard, mouse only make sense in relation to the workstation they are part of. This also means the purpose and use of the informational objects enjoy an interplay with the purpose and use of the physical objects they are embedded in. Gersbo-Møller and May[41] have used inquiry into this interplay to form a design approach for pervasive systems based on affordances.

B.viii Convergence of pervasive computing

The realization of Weiser’s vision is expected to happen gradually. The initiating hypotheses are stated in reference to this vision; this means that the degree to which they hold true is expected to increase alongside the gradual realization of the vision.

We say that pervasive computing is a convergent technology, and in more ways than one. The first is that the technology is imagined to converge towards Weiser’s vision, starting from crude experiments and gradually approaching the vision for each successive implemented system. Part of this is also, as pointed out by [19], that different technologies enable pervasive computing: Processor technology by converging to smaller and more powerful devices, batteries becoming more effective, network technologies becoming more advanced (ad hoc networking, [52]), increasing bandwidth.

The second way to speak of convergence of pervasive computing is at a more nontechnical level, since the emergence of fully integrated sociotechnical systems [36] will create new research areas and blend old ones, for instance organizational theory and software engineering. A “sociotechnical system” is a term for an information system made up of a pervasive system and, for lack of a better word, an organization, the two blended together and interdependent enough that it does not make sense to consider either as an isolated system.

Huang [25] has even observed a convergence between architecture of buildings and the virtual world, and advocates that the these two worlds should be designed to complement each other.
C Delimitation and Formal Goals

This section argues and states the formal goals in terms of the concepts and analysis in the previous section.

C.i Modeling and Methodology for pervasive computing

The first initiating hypothesis of the project stated that “the analysis and design methods and modeling tools used for today’s administrative software systems are insufficient for building pervasive systems.” (page 2).

Kristensen et al.[40] supports the claim that making a system pervasive creates a significant impact on system analysis and design. Further, “modeling concepts ... have transcended programming languages and have been used as a common way of understanding and describing software, hardware, organizations ...” [40]. This shows the importance of creating foundation of modeling concepts that the software for pervasive computing can be build upon, c.f. the Scandinavian tradition of conceptual modeling [34]. This leads to the formal goal on the subject of the above hypothesis, which is to investigate:

- What conceptual models and approaches are necessary to improve current analysis and design methods to better enable development of pervasive systems?

C.ii The influencing forces on software

The second initiating hypothesis stated that “for a software system and in particular a pervasive one, the system's alignment with the organizational and social situation in its context is crucial.” (page 2).

As motivation for this hypothesis it has already been stated that this alignment is not always present because of failure to address the soft issues during system development [45, 44]. Thus it is not uncommon to hear about administrative systems that has to be “worked around” [60, 18, 61]. This must mean that the development processes used to build these systems can be improved in this respect. Such an improvement must be based on knowledge about how software systems and organizations should relate. For instance, a relationship may be that the organization of the users require the system to exhibit particular characteristics. This could be a system used at a hospital, where the nature of the data held by the system (personal information) impose a requirement of security on the system.

In this project particular focus is on the general kind of relationship, where the nature of the organization of users impose requirements of some qualities in the system. The choice of this focus and the approach in the planned activities is based on a general view of an organization as shown in figure i.3. The figure illustrates how the structure of an organization emerges as a result of a large number of different forces.
Figure i.3: The influencing forces forming the structure of an organization.

Software used in organizations is also subject to a range of influencing factors dictating the choice of functional and nonfunctional requirements, and later how and if the software is used. Software systems and organizations have a number of things in common, and both are subject to a number of the same forces (e.g. economy, culture etc.) although they may affect software and organizations differently. This implies that the forces that are important to the shape of one may also be important to the shape of the other. The point of this is that some of the theory of organizations, which deal with their structure and forces, can be useful in relation to creation of software.

As per the discussion of design philosophies, they are means of achieving particular qualities in the developed system. Thus, on the subject of the second initiating hypothesis the question to investigate is:

- What is a suitable relationship between a system and its affiliated organizations and how can knowledge about the structure of these be used to select design philosophies that will help achieve such a suitable relationship?

Apart from selecting design philosophies, if it is known what qualities in a system are desirable for a given organization, this can be applied in the prioritization of architectural attributes in methods such as ATAM[31].

C.iii Formal Goals

For easy reference, the formal goals are repeated here:

- What conceptual models and approaches are necessary to improve current analysis and design methods to better enable development of pervasive systems?
• What is a suitable relationship between a system and its affiliated organizations and how can knowledge about the structure of these be used to select design philosophies that will help achieve such a suitable relationship?

D Approach

Five activities was undertaken to investigate the questions stated previously. Activity number 6 was not initially planned, but still serves the purpose of the first formal goal about modeling and approaches. The activities were both practical (I, II, V) and analytical (III, IV, VI). The activities are briefly summarized below, but the method and purpose of each are explained in more detail in the individual chapters.

I. The first activity dealt with development of an administrative system for an academic department of a university, the Maersk Institute. It was made using the Unified Process, and the workflows of Business Modeling, Requirements Capture, Analysis and parts of Design were performed.

The purpose was to introduce the Unified Process and get familiar with the problem and application domains as a preparation for the next activity. A more detailed description of the activity and a summary of the development is given in chapter 1. The complete documentation of this development is placed as appendix D to this report.

II. In this activity the system considered previously were extended to become pervasive. The same workflows of UP were performed and the purpose was to observe what difficulties were encountered either with doing the prescribed workflows or representing their results in the Business, Requirements and Analysis models. Based on these difficulties a number of criteria were stated as the requirements to the modeling technique proposed in activity four. As for activity I, the complete development report has been placed in an appendix of this report, appendix E. The observations, analysis of them and a set of criteria is placed in chapter 2.

III. This activity consisted of two parts. Firstly a comparison of software and organizations was made, with the purpose of justifying the assumption that both are subject to some of the same forces. Secondly, each of Mintzbergs five stereotypical organizations were analyzed and the forces, as identified by Mintzberg, were considered to derive the effect these forces have on a software system. This lead to a proposition of design philosophies for four of the stereotypes. The activity is documented in chapter 3.

IV. A review of the philosophy of modeling formed the base for this activity, and was used together with the criteria from activity two to propose a modeling technique for pervasive systems. This work is presented in in chapter 4.
V. This activity is about testing the modeling technique resulting from activity VI. It is documented as the examples in chapter 4.

VI. In this activity the pattern Interaction Widget was described. The description is chapter 5.

The dependencies among the five activities are shown in figure i.4.

E Results

This section presents the results of the project in the form of a number of hypotheses. They are repeated immediately below for easy reference, and each is argued in the following subsection.

**Hypothesis 1** The nature of the influence of the organization of the users is orthogonal to a system being pervasive. But the influence is more important because the impacts of mismatch between organization and system are more powerful.

**Hypothesis 2** The forces that shape the structure of an organization also have an effect on the software that is used in this organization.

**Hypothesis 3** Some design patterns traditionally used only in the informational dimension of program design can be extended to include the physical dimension.

**Hypothesis 4** A pervasive system pervades our world, but the reverse is equally true, that the world pervades the system.

**Hypothesis 5** In the Unified Process, the workflows business modeling, requirements capture and analysis cannot in their current form be carried out entirely as prescribed when the system under construction is pervasive.

**Hypothesis 6** A map depicting the spatial and locational aspects of a pervasive system in terms of areas, tangible objects, points, physical objects provide a necessary frame of reference for reasoning about these aspects of a pervasive system.
Hypothesis 7 Minimizing the number and severity of occurrences of breakdowns in the interactions with pervasive technology is a necessary goal for making this technology disappear.

Hypothesis 8 The Unified Process and the format (metamodel) of its prescribed requirements model does not support the identification and representation of the system qualities peculiar to pervasive computing.

Hypothesis 9 Based on the structure of the organization of the users of a system, it is possible to identify certain qualities should be prioritized for this system and its parts.

Hypothesis 10 There are at least the following four categories of relationships between a system and the organization of its users.

Hypothesis 11 The notion of a system implicitly assumed in the Unified Process is not adequate for describing pervasive computing.

Hypothesis 12 The paradigm employed in multi agent systems is one way of dealing with the internal-external modeling problem of tangible objects.

E.i The arguments for the hypotheses

HYPOTHESIS 1
The nature of the influence of the organization of the users is orthogonal to a system being pervasive. But the influence is more important because the impacts of mismatch between organization and system are more powerful.

The nature of the influence of the organization of a system’s users is hypothesized orthogonal to the system being pervasive because this influence, as presented in chapter 3, is largely independent of a system being pervasive.

A pervasive system can be said to have a stronger presence in the life of its users. This is because the presence is not limited to the time the users spend in front of a workstation.

The mismatches between a system and the organization of its users often reveal themselves through low acceptability. The stronger presence of a pervasive system increases the degree to which a mismatched system can bother its users. Therefore a careful match between a pervasive system and the organization of its users is important.

HYPOTHESIS 2
The forces that shape the structure of an organization also have an effect on the software that is used in this organization.

The analysis in chapter 3 provides various examples of such forces. The conclusions that are made based on analysis of these forces are supported by empirical
findings documented in the literature referred to throughout chapter 3. This is interpreted to yield support to the hypothesis. It is important to note here that the hypothesis makes no claim that all forces influencing the organization necessarily are important to software, nor vice versa.

An example of a force affecting both system and organization can be found in Adhocracies. The nature of the work performed in these requires a lot of communication among people educated in different disciplines. The organization is structured to support and enable this communication (team and project based work units, informal). Studies such as [48] shows that communication technology is important in adhocracies (especially if spatially distributed). So the nature of work performed acts as a force on both the organization and the system towards prioritizing communication.

**HYPOTHESIS 3**

*Some design patterns traditionally used only in the informational dimension of program design can be extended to include the physical dimension.*

The *Interaction Widgets* pattern described in chapter 5 is an example of such a pattern. It can be used in the informational dimension that traditional software design is constrained to, e.g. it is applied in the concept of buttons in a GUI. But as is evident from the application of the pattern in the *Context Toolkit*[28] it can also stretch across both the informational and the physical dimension like pervasive systems do.

Whereas this project has mainly been about the differences between traditional and pervasive computing this hypothesis is interesting in that it provides an example of a kind of knowledge about software design which can be reused.

**HYPOTHESIS 4**

*A pervasive system pervades our world, but the reverse is equally true, that the world pervades the system.*
A general system architecture is shown in figure i.5, as proposed in [38, p. 10]. The division of interface, control and model parts of a system corresponds to the stereotypes boundary, control and entity classes used in the Unified Process. In a client server system, [38] lists four patterns for distribution of interface, control and model parts of a system. These are illustrated in the middle of figure i.5, where the angle brackets around a letter denotes optional placement. For instance a centralized system with thin clients would have Interface at the top and Control, Model at the bottom, meaning that everything but the user interface resides on the central server.

The model component used in figure i.5 is “the part of a system that realizes a model of the problem domain.” [38, p. 230]. In pervasive computing, the tangible objects are part of the problem domain. This merger of informational model and that which it models has been expressed in the definition 4.1 of a tangible object, page 66. According to this definition of a tangible object, the object in the informational sense and the object in the physical sense shares one identity. This is different from traditional systems, where an informational object such as a book always has an identity separate from the identity of the physical book it represents. A consequence of the merger between an informational object and the physical object it models, is that the informational objects in the model part of the system are necessarily distributed as much as the physical objects they represent. This is shown to the right in figure i.5 by making the distribution of the Model fixed rather than optional. It may still be that parts of the model are centralized, (e.g. bank accounts are likely not tangible).

From the perspective of the users, we build computation into the physical objects. But from the developers perspective, it is the properties of physical objects that pervade the informational model (e.g. the distributed nature of tangible objects).

HYPOTHESIS 5
In the Unified Process, the workflows business modeling, requirements capture and analysis cannot in their current form be carried out entirely as prescribed when the system under construction is pervasive.

The observations presented in chapter 2 shows what kind of concrete problems prevented the prescribed application of UP in the construction of a pervasive system. They were:

1. No suitable concepts for modeling tangible objects, since neither classes (internal) or actors (external) seems adequate.

2. Problems with analyzing and designing the physical aspects of a pervasive system. This is both lack of means of expression in UML, and of prescribed ways of dealing with these physical aspects in UP.

1Translated from Danish.
3. The business model that UP prescribes does not provide an appropriate base for identification of use cases for a pervasive system.

4. Lack of means to specify and handle the requirement that it should be possible to access use cases through different user interfaces.

It should be noted that main parts of the process remain meaningful, including the perspectives employed in the workflows treated in this project.

**HYPOTHESIS 6**

A map depicting the spatial and locational aspects of a pervasive system in terms of areas, tangible objects, points, physical objects provide a necessary frame of reference for reasoning about these aspects of a pervasive system.

In chapter 4 a format for such a map was proposed. This was done based on the criterion 2.2 page 43 which specified the need for a frame of reference for the physical aspects of a pervasive system. Example 4.2 page 71 shows how a map in this format provides such a frame of reference by defining e.g. particular areas and positions of physical objects (both “dead” and tangible).

**HYPOTHESIS 7**

Minimizing the number and severity of occurrences of breakdowns in the interactions with pervasive technology is a necessary goal for making this technology disappear.

When pervasive computing is envisioned to disappear, this does not mean that we do not know it is there, but that it does not draw attention to itself.

By the nature of breakdowns as defined in 4.6 page 73 they are interruptions in the flow of attention between the primary tasks we engage in. They describe the situations where the technology (or some other entity) draws attention to itself.

**HYPOTHESIS 8**

The Unified Process and the format (metamodel) of its prescribed requirements model does not support the identification and representation of the system qualities peculiar to pervasive computing.

This is to say that a system created with UP does not possess this certain “smartness” that is envisioned for pervasive systems. One reason for this, as far as can be supported by the findings in this project, is that the qualities of adaption, anticipation etc. are not specified in the requirements model.

This is because if a pervasive system is built (using the modeling techniques proposed in chapter 4) with the sole specification of a use case model then it will not exhibit “emergent behaviour” in the sense of being able to participate in new use cases (interactions) c.f. the dynamic property in the table in appendix C.

**HYPOTHESIS 9**

Based on the structure of the organization of the users of a system, it is possible to identify certain qualities should be prioritized for this system and its parts.
For instance, if the system is to be used in a professional bureaucracy, acceptability is more important than in a machine bureaucracy. If the system is to be used in an adhocracy, then the communication facilities are of paramount importance. If the system is used in a machine bureaucracy, the handling of business rules should be made as flexible as possible.

In each of these organizational types a different part of the organization is dominating. The qualities highlighted for each of these are those important to the dominating part. Thus in a given system, the flexible handling of business rules is important for the parts of this system used by users that are in the technostructure of the organization of the system’s users.

**HYPOTHESIS 10**

*There are at least the following four categories of relationships between a system and the organization of its users.*

1. Consider the case where the system and the organization are linked by similarity. This could be the case if they are parallel in structure when each is seen from the same perspective, or if processes occur in the same manner in the system and the organization.

   For instance, in [26] a hospital system is build as a multi agent system that has an agent for each person at the hospital, mirroring this persons responsibilities within the system, acting on behalf of this person. Also, the system nodes are organized in a tree structure such that their deployment match the hierachical division of hospital into departments, subdepartments and rooms. For static organizations this relationship seems advantageous with respect to evolution.

2. Another way they might relate is by direct dependency of one of the two, on events occurring in the other. For instance, if a meeting is moved, a pervasive system that incorporates calendar functionality should know about this (a dependency of the pervasive system on the organization). When the system holds this information, it may have to communicate to whom it may concern in the organization, that the meeting room is free on the date and time in question.

3. As in the relationships investigated in this report (c.f. the last hypothesis) dependencies might also be more subtle, so that the type of the organization with which the system is affiliated, require the system to exhibit characteristics that complement this particular type of organization. For instance, if the system is used at a hospital, the nature of data used at hospitals (personal information) impose a requirement of security on the system.

   Note that what is illustrated in this example can be seen as a kind of similarity. The organization (the hospital) must have a quality of security, among other things to protect the patient records. This includes doctors and nurses
needing to take a vow of silence, and the patient records to be locked away.
A system that manages electronic patient records at a hospital needs to have
similar qualities, and as such might be seen as similar to the organization in
this respect. In the same way, it is important for both to be reliable.

4. Finally it may be the case that there are several opposing forces shaping the
relationship, in which case it would seem suitable to describe it in the form
of patterns. The idea of using patterns to describe the relationship between
a system and organization is not new: [12] describes many such patterns,
but they are about the relationship between the systems architecture and the
organization of its developers. What is proposed here is that the relationship
between the system and the organization of its users can be described with
patterns.

An actual relationship may contain elements of all three types.

**HYPOTHESIS 11**
The notion of a system implicitly assumed in the Unified Process is not adequate
for describing pervasive computing.

In the Unified Process, as system is “A collection of subsystems organized to
accomplish a specific purpose and described by a set of models, possibly from
different viewpoints.” [27, p. 432]. The problem with this definition is that a given
entity is either part of the system or part of its environment (external to the system).
The observation that fostered the concept of a new stereotype for tango-classes
were that a tangible object could be suitably modeled neither as an actor nor a
class. These are by definition internal respectively external to the system.

A consequence of this problem was observed as a lack of control over the
objects if they were considered internal. On the other hand, if they are considered
external then there by definition cannot be a system made of only tangible objects.

**HYPOTHESIS 12**
The paradigme employed in multi agent systems is one way of dealing with the
internal-external modeling problem of tangible objects.

In multi agent systems [5] the notion of a system is more open-ended by nature
since such a system is made solely of agents. That an agent can come and go (like
in organizations of humans) fits with the property of tangible objects that they can
enter and exit a “system” at will (of some human who makes this happen). One
can say that in these “systems” the coupling among agents is so loose that external
vs. internal is not an issue.

A further point about multiagent systems is that emergent behaviour is ex-
pected, so this alone makes the paradigm more suitable for achieving the proper-
ties in table C.

Finally, since agents by definition are conceptually closer to humans—implicitly
they possess some degree of animacy—the similarities between the organization of
software and of humans gets even more numerous. In fact, moving from traditional systems to multiagent systems it may just be correct to say that software "organizations" converge towards organizations of humans. Since convergence of research areas is one of the properties noted for pervasive computing this is not unexpected.

F Conclusion

This project as a whole has touched upon a wide array of subjects—from organizational theory over philosophy to concrete class diagrams of program design. The subjects addressed are united by their relevance to the development of software for pervasive computing. A development process and modeling language tailored to pervasive computing is not created by a single effort, but rather evolves alongside pervasive computing converging towards its envisioned ideal of disappearance. This project has addressed some of the issues that should be handled by such a process, and it has resulted in outlines of some building blocks that are needed to model a pervasive system.

F.i The Role of Pervasive Computing

The properties and nature of pervasive computing have already been discussed elsewhere, but the more profound question of what it is good for [24, 64] was not initially planned to be discussed in this project. However, the proposed approach to break-down centered business modeling and identification of use cases implicitly address this question. The motivation of this approach is that by avoiding the kind of breakdowns that are undesirable, technology may not just become easier to live with, it may even disappear. So this approach is based on the view that the role of pervasive computing is to make our lives easier.

F.ii Experimental and Analytical methods of work

The work in this project has been both of experimental (activities I and II) and analytical (activities III–VI). Mathiassen and Stage [39] describes the relation between these two methods of work in The principle of limited reduction. “This principle states that relying on analytical behavior to reduce complexity introduces new sources of uncertainty and this requires experimental countermeasures. Correspondingly, relying on experimental behavior to reduce uncertainty introduces new sources of complexity requiring analytical countermeasures.” [39].

The principle is descriptive of the work in this project. At its inception there were little to back up the initiating hypotheses; a situation characterized by uncertainties. The activities I and II reduced these uncertainties—the first activity increased the amount of information about the problem and application domains of the Maersk Institute, and the second activity increased the amount of information about pervasive computing in relation to these domains.
To make sense of the information thus generated in activity I and II (that is, to reduce the complexity stemming from too much information) the work in activity IV used an analytical approach to propose a modeling technique. The uncertainties characterizing the proposed modeling technique—it had not been verified—were then to some degree reduced in activity V.

Activity III about the relationship between a system and an organization were analytical, because what it effectively set out to do was make sense of the abundance of information (and thus following complexity) that relates to this subject and is already contained in the articles referred to throughout chapter 3.

In relation to pervasive computing, the analytical activities III–VI has increased the insight into this subject, and allowed formulation of the set of hypotheses in section E, the results of the project. This indicates that to continue the line of work in this project, an experimental method of work is expected; this is discussed in the next section.

F.iii Future Work

Test of Modeling Technique

The modeling technique proposed in chapter IV has not been tested in a full-length development project. To develop a system such as the one considered in chapter two over several iterations requires an amount of work that sets it outside the scope of this project. Using the modeling techniques in such a project would be the next step in refining them, as per the principle of limited reduction.

Relationship between System and Organization

Future work on the subject of the relationship between a system and the organization of its users could be either to investigate other kinds of relationships as proposed in hypothesis 10, or to try applying the results of chapter 3. The argument to try these in a development project is not so much to verify the soundness of the matches, because these are fairly well supported by empirical studies. Rather it would serve to reduce the uncertainties related to the form they are proposed in, and the suggested way of achieving them through use of design philosophies.

Agent Oriented Development

In relation to hypothesis 12 it would be interesting to further explore the suitability of agent oriented development for pervasive systems. Especially in that it makes organization of software more alike to organizations of people, and thus in theory enforce not only the convergence of these two towards fully integrated “socio-technical” systems, but at the same time enforce the view of them as being shaped by some of the same forces.
Modeling “Pervasiveness”

Hypothesis 8 stated that the current format of requirements models are not targeted at specifying the qualities of anticipation, adaption etc. that pervasive computing should have (c.f. the table in appendix C). It could be interesting to investigate not just how to capture requirements of this nature in a suitable model, but also how to model them internally in the design of the system.

Patterns for pervasive computing

A pattern that can be used in pervasive computing was given in chapter 5, but there are still a number of interesting questions relating to patterns for pervasive computing. Firstly it would be interesting to find examples of patterns whose problems were characterized by forces some of which were opposed despite being in different dimensions (physical, informational, organizational).

It would also be interesting to look for patterns of relationships between as system and the organization of its users.
Chapter 1

Applying UP to the Maersk Institute Administrative System

This chapter describes activity one in the array of activities selected in the problem analysis. The chapter starts out with a description of the purpose of the activity, in section 1.1. The method used to achieve this purpose is argued in section 1.2.

The result of the activity are three models, the format and purpose of each is described in section 1.3. Section 1.4 presents the practical work, including the concrete models, that is, Business Model, Requirements Model, and the Analysis Model (henceforth abbreviated BM, RM and AM, respectively). Section 1.5 contains the secondary results of this work, namely what has been made of observations pertaining to the experience of applying UP. The chapter closes with section 1.6 that concludes on this activity.

1.1 Purpose

The purposes of this activity are the following:

- Most importantly is to establish a base for the activities that are to follow.

- The form used to obtain the above goal is a practical development project. This enables for observations to be made about the process that is applied, namely the Unified Process. Conforming with the overall scope of this msc project, modeling will be the focus of attention for these observations.

1.2 Method

The practical work in this activity is to develop an administrative system for scheduling meetings, reserving resources, and other administrative tasks at the Maersk Institute. Rather than “the system” this will be referred to as TAIS (the Maersk
Institute Traditional Administrative System). The word *traditional* is here used as opposed to *pervasive*.

The work has been carried out as described in the chapters 6–8 in [27]. The results of each step in the work is placed in a separate document, which located in appendix D. Refer to this for the full models, as only an overview is given here.

### 1.2.1 Focus on Business, Use Case and Analysis Models

It is believed that some parts of UP are more fruitful to work with than others. Specifically the project focus on business modeling, requirements, and the analysis workflows, for these reasons:

- Capturing and Modeling the requirements is prioritized because the means by which these activities are achieved says a lot about what kind of systems we would like to build. If a system should be truly pervasive, then this is (obviously) a requirement, and should therefore be incorporated in the requirements model. But is the current format of requirements models able to capture this quality of *pervasiveness*? Being able to express this quality would increase the understanding of pervasive computing, which is one stated purpose of this project.

- If the answer to the above question is no, that pervasiveness cannot be expressed using the current kinds of requirements models, it still makes sense to perform analysis. The reason is pervasive computing’s property of being convergent. When developed systems does not become pervasive from one day to the next, it is certainly expectable that development methods and modeling does not change overnight either. Thus, even if the format of requirements models is not perfect it will likely allow for some sensible information to be contained in the models. For instance, though a use case may not quite capture the adaptive quality of a system, it will contain some truth about an actual usage scenario. So it is expected that some inabilities of the format of an analysis model may be uncovered, e.g. lack of means to reason about the physical nature of a pervasive system.

- Business modeling is relevant to this project because it is the activity where a model of the organization of the users is used as a basis in the system development. Focusing on this activity thus provides a base against which to argue that the findings about the system vs. organization relationship differ from existing knowledge. In relation to modeling pervasive systems this activity is important because it is the model from which use cases are identified. For traditional systems it is thus an aid in specifying what the system should do.

This states why some stages are important, but not why design, implementation and test are of lesser importance.
1.3 Three classes of models

The Unified Process describes a series of activities that leads to a number of models. The models each reflect different views on the system to be build, according to what is in focus in requirements capture, analysis, design etc.

This activity must provide a base for the second activity of making the system pervasive, so an obvious requirement has been to become familiar with the context of the system, which is done through the development of BM, RM and AM. The purpose and contents of each of these are described in the following subsections. The following descriptions of the models are extracted from the chapters 6–8 in [27].

1.3.1 Business Model

PURPOSE

The purpose of BM is to describe the business processes taking place in the context of the system. Thereby it clarifies what value the business creates for its actors, and thus helps ensure that the right system is being build. In one view it describes how things are done before the system is taken into use. For instance, BM might include a description of how resources are reserved, a task which will be carried out differently once the system is used to accomplish it.

CONTENTS

The BM consists of a business use case model and a business object model. Each business use case in the business use case model shows how the actors of the business uses the business to obtain some goal. The business object model describes the entities of the problem domain, and how they are used by the actors to realize the use cases.
1.3.2 Requirements Model

PURPOSE
The RM in Unified Process is a use case model. It serves as an agreement between
the customer and the developer. It describes, in use cases, how the system and its
users interact. Finally, it is the primary input to create the AM, and also used as
input to the design- and test- models. As opposed to the business use cases, the use
cases of RM show how things are done after the system is taken into use, that it,
how a task is performed using the system.

CONTENTS
RM contains use cases, actors, an architecture description, a prototype of the user
interface, and a glossary. Here a use case may be subject to non functional require-
ments (e.g. performance, reliability). It describes, step by step, how actors use the
system to accomplish some task.

The architecture description prioritizes the use cases according to their archi-
tectural significance, which is done according to what risks are associated with
them.

The prototype of the user interface allows the users to give some feedback on
the initial appearance of the system, and whether it is what they have in mind. The
glossary defines common terms.

The BM is use as input when identifying the use cases of the system.

1.3.3 Analysis Model

PURPOSE
The analysis model is the first model of the internals of the system, and to allow
reasoning about this it uses the language of the developers. One of its purposes is
thus to provide a first cut at the design model, though it is less detailed, and inde-
pendent of the technical platform. It also serves to further specify the requirements
given by the use case model.

CONTENTS
The analysis model consists of analysis classes, use case realizations–analysis,
analysis packages, and an architecture description.

The analysis classes come in three stereotypes. The entity classes represent
some data, and some of them can often be traced from the entities in the business
object model. The control classes are the active classes; they implement the busi-
ness logic, and often controls the flow in a use case realization. The boundary
classes are those that interact directly with an external actor.

The use case realizations–analysis show how the analysis classes interact to
realize the use cases of the RM. Each contains a class diagram depicting the par-
ticipating classes, a collaboration diagram showing the dynamics, and a textua
description of each step in the activity diagram. Nonfunctional requirements may also be associated with it.

The architecture description draws out the architecturally significant analysis classes (e.g. control classes managing realizations of architecturally significant use cases, entity classes representing central entities from the problem domain, analysis classes with many relationships to other classes), and shows how the analysis classes are distributed onto analysis packages.

1.4 BM, RM and AM for the Mitas

What has been performed is the first iteration, which thus belongs to the inception phase. In section 13.4 in [27] it is discussed which activities within UP are of relevance in the inception phase. The work here is carried out conforming with this, e.g. a prototype of the UI has not been made, but relatively more work has gone into identifying the use cases so as to scope the system. The goal of such a first iteration thus matches the purpose of this activity.

1.4.1 Business Model

This describes the business use case model and the business object model for TAIS. The business object model should make apparent how the business use cases are realized by actors and entities of the business. This has not been done, since the step-by-step descriptions of the use cases are exhaustive. The two following models thus describe the internal and external of the business at the same time, as opposed to how RM and AM describes the system from an external and internal perspective respectively.

Business Use Case Model

The relations of the actors and the business use cases are shown in figure 1.1.

The Secretary may use the Arrange Board Meeting use case to call for points to the agenda of the meeting, when a board meeting has already been scheduled using Schedule Meeting. The Arrange Board Meeting requires the assistance of several Officials, who are the participants in the meeting and thus has the right to submit points for the agenda.

Turning to the use cases that can be initiated by an official, the Schedule Meeting is used to find a date and time for a meeting when it’s participants are known. The Reserve Projector and Reserve Meeting Room are used by an Official to reserve a resource, for a given period of time at a given date and time.

1Exhaustive in the sense that they describe every step, both internal and external. This is done because the actors are all internal to the business, which means that they interact directly with the business entities. For that reason it is considered unnatural to divide the view of the use cases into an internal and an external one.
Figure 1.1: The business use cases and actors in BM for TAIS.

Figure 1.2: The use cases and actors in RM for TAIS.

**Business Object Model**

The objects in the business object model and their associations are shown in figure 1.2. The Calendar is a physical book into which reservation entries are entered. The Projector is one of the resources that may be reserved by entering an entry in its Calendar, as is the Meeting Room. A Meeting is associated with precisely one Meeting Room, which is the place where it is held. A Meeting Room may of course be associated with many Meetings, as long as they are scheduled for different times, a constraint that is ensured by the Calendar for the Meeting Room.

A Business Rule is more abstract than are the other entities. A definition of a business rule is “Statements that constrain, derive, and give conditions of existence. Business rules are used to specify allowed state of affairs, including allowed
A business rule may be written down as a formal rule, or it may be more tacit, if there is a common but unarticulated agreement about it. As an example, suppose that some students have reserved a meeting room with the intend of watching a movie there. If it later turns out that an exam should take place in the room at the same time, it seems sensible that the first reservation are overridden. A business rule can be used to capture when reservations can be overridden. To generalize, a business rule will in this context be characterized by the persons, groups of persons, or entities that it is concerned with, and how these relate to some action. For instance, it may state that a professor may override the reservation of a projector when it is made by a student. Here the concerned groups of persons and entities are professors, students, and projectors. The action is to override a reservation.

### 1.4.2 Requirements/Use Case Model

The use cases, actors and their relations are shown in figure 1.3. (Reserve Meeting Room and Reserve Projector are only shown as illustration of how Reserve Resource is derived from the business use cases; they seem similar enough that any distinction is superfluous).
Starting with Arrange Board Meeting the secretary may use this use case to call for points for the agenda of the meeting. As for the corresponding business use case, this can only be performed once the board meeting has been scheduled using Schedule Meeting.

Moving on to the ordinary User, any use of the system by this actor (and others who are specializations of it) starts with logging on to the system by means of Login, and ends by logging off the system, with Logout. The User may now use Schedule Meeting to find a time and date for a meeting where a suitable number of participants will be able to attend. (By Suitable is meant the clarified in the next paragraph). Once a meeting has been scheduled, and the user in question is invited (this happens as part of Schedule Meeting, where each participant is notified of the time and place for the scheduled meeting), the User may receive a request for submission of points to the agenda. Performing this submission can be done in Submit Agenda Points.

The System Administrator will use the system to manage accounts, creating/editing and deleting them with Create/Edit Account and Delete Account respectively.

Finally, the Manager has the right to change and add business rules using the Create/Edit Business Rule, and Delete Business Rule use cases.

The similarity in the flow of events between the initial Create/Edit Business Rule and Create/Edit Account led to the more general (abstract) use case Create/Edit Entity, from which they both inherit. The same is the case for the use cases concerning deletion of entities.

1.4.3 Analysis Model

The distribution of analysis classes onto analysis packages are shown in figure 1.5 page 34.

As an initial identification of packages two has been found, namely one application specific and one application general. The classes with a fill colour are those in the application specific package, whereas those that are white are in the application general package. Note that the purpose of this diagram is to visualize the class-level dependency among the two packages. What can be noted about this is that the application general package is dependent on the application specific one in only the case of Account having to know of Calendar. This is a problem because something general is only generally applicable if it is not dependent upon the more specific parts of the application. This problem will be dealt with in design, to make the application general package totally independent of the application specific one.

1.4.4 Design Model, partly

The work on carried out on the analysis model has given an increased understanding of the system. The division and identification of analysis packages was performed in the start of the analysis workflow. The first activity in the design work-
flow is to give an initial identification of design packages. This initial identification can be made on grounds of the experience gained during analysis, so it seems in order to present this first part of the architectural design here as well. The packages identified in the architectural design are shown on figure 1.4.

Relative to the analysis packages, the packages accounts and brules are traceable to the analysis package Accounts/Business Rules Handling, that is, the application general package. The mitas package and entityeditor packages are traceable to the Meeting/Resource Handling package (application specific). Note that functionality now located in the entityeditor package has moved from the application specific to the application general layer. This is because it is evident from the analysis model (specifically the use case realizations—analysis), that entities such as Resource, Meeting, Account, Business Rule all have in common the way they should be created/edited/deleted through the various user interfaces. This commonality might as well be extracted for several reasons. Firstly it will help create a more uniform user interface, secondly it will remove some redundancy in gui classes, thirdly it will make it easier to add gui support for new entities, an finally the resulting subsystem may be reused across applications.

1.5 Observations made about UP and development

It is apparent that the UP, as described in [27], has a set of rather formally specified activities within both requirements, analysis and design. That the B-, R-, A-, and D-models affect each other is also incorporated into the process, by letting RM be input to the activities that result in AM, and so forth.

The observations made are focused on how the models affect each other. To be able to easier formulate the observations, the following two concepts are defined.

**Definition 1.1**

The primary use cases are those implementing the functionality that is the reason
for building the system. They are often traced directly from the business use cases.

**DEFINITION 1.2**

The *secondary use cases* are those that support the primary ones. For instance when having a primary use case such as Schedule Meeting, it is apparent that there must be a number of supporting use cases for creating, editing, and deleting meetings. These are secondary use cases.

### 1.5.1 Observations pertaining to BM and RM

The following kinds of changes was made to the BM of T AIS. For each kind of change in BM, the effect in RM is noted immediately below, in *italic* type.

1. Addition of a use case. In the sense that this is what happens whenever a new iteration is started, this is the obvious choice of change, but also one that would be expected to be incorporated in the method.

   *In every case these changes caused the addition of a primary use case, and one or more secondary use cases. But the functionality of the primary use cases were apparent from the business use case. The functionality of the secondary use cases were derived from the entities that were used in the business and or primary use case.*

2. Identification of similarities in use cases. This corresponds to the micro iteration described in example A.1. In other words it is generalization across multiple business use cases.

   *This was done in one case, described in example page 83. Here two business use cases were generalized. The effect this had on the use cases in the requirements model was for two of them to be merged into one. Both of these were primary use cases, so there were no changes in the secondary use cases of the requirements model.*

3. Change in the set of actors. This refers to what happens when new business actor types are added and when existing ones are re-arranged with respect to the roles they describe (for instance, letting a meeting be arranged by a Secretary instead of an Official.)

   *At some points it was changed which actors initiated which use cases. When the two actors of such a change each has an equivalent in RM, it was noted that an equivalent change occurred in the primary use case traced from the business use case in question. The secondary use cases were only in some cases changed as well.*

### 1.5.2 Observations pertaining to AM

A number of shortcomings of the system under development were detected during analysis stage of the first iteration. These were:
1. How a message gets to a user, browsing of messages by a user etc. In other words, there are no use case in which a user actually reads a message, although there are use cases where messages are sent.

2. That a user should be able to send a “free-form” message to another. This is actually to ensure that the system can have an, although limited, possibility for being applied to tasks it was not specifically designed for.

3. It should be possible to create/delete/edit Resources.

4. It should be possible for a User to edit his or her calendar.

5. Deletion/Editing of meetings.

1.6 Conclusion

There were two goals of this activity.

- It was meant to establish a base for the following activities. This has been done by:
  - becoming familiar with the problem domain.
  - setting the scope for what areas of work should be supported, namely managing meetings and resources as well as time management for employees.
  - exemplifying the use of the Unified Process, thereby presenting something concrete for the development in activity two to be held up against.
  - introducing the way in which UP incorporates information about the organization and problem domain into the system development (e.g. the Business Model).

- The second goal was to make some observations about how the models BM, RM, and AM affected each other. This has been done, and the observations were noted in section 1.5. These observations are discussed in appendix A.

The activity thus seems to fulfill its purpose of playing an enabling role for the remaining activities.
Figure 1.5: The analysis classes and their relations. The classes with no fill colour are in the application general layer.
Chapter 2

Deriving criteria for a modeling technique

This chapter describes the second activity which is about extending the TAIS system so as to become pervasive. The Pervasive edition of TAIS will be called PAIS (for Pervasive Administrative Information System).

Section 2.1 describes the purpose of the activity. Section 2.2 explains the method, shaped around a development case, used to achieve this purpose. Section 2.3 presents the actual development case in terms of the models it resulted in and the observations it gave rise to. The complete documentation of this development can be found in appendix E page 97+37. Section 2.4 contains an analysis of the case and based on this analysis a derivation of the set of criteria for a modeling technique. Section 2.5 sums up and concludes this activity.

2.1 Purpose

The purpose of this activity is to obtain a number of criteria by which a modeling technique for pervasive systems can be judged. These criteria will function as a set of requirements for the modeling technique. They will also enable a concrete technique to be evaluated.

2.2 Method

The method through which the set of criteria will be derived is build around an experiment.

The knowledge used to build traditional software systems include process descriptions, modeling techniques and a set of concepts around which the first two are centered. The concepts include architecture, classes, objects, associations, patterns. The general idea is to conduct an experiment wherein this knowledge is applied in the development of a pervasive system. It consists of extending the TAIS
to become pervasive, and is described in section 2.3. The choice of process was UP as for the first activity. The modeling was done in UML as incorporated into UP.

There is thus an expectation that the experiment will issue forth a number of concrete but unspecified observations pertaining mainly to the format of requirements and analysis models. Apart from these concrete observations, the system that is the product of the development is held up against the characteristics of pervasive systems, as put forth in the explanation of pervasive computing page 7. This is done to evaluate how pervasive the developed system is, and thus whether application of traditional development methodology results in systems that are truly pervasive in flavour.

2.2.1 Consequences of extension instead of a new system

This section discus the matter of extending a system versus starting from scratch, and how this matter has been handled.

TAIS was not a complete and running system, since only BM, RM and AM exists for it. This means that the impact of it is smaller, because it is easier to modify these models than it is a fully implemented system.

During the work with making the system pervasive, the existing body of work that is TAIS, is regarded not so much as a platform to build upon as a legacy system that can be encapsulated behind some interface, and from which some functionality can be reused. This view is a consequence of what is known about TAIS, mainly what use cases it provides and from AM what its data model contains. The view is also consciously chosen, because adapting it serves to limit the constraining effect of TAIS on the shaping of PAIS. It is of interest to limit this effect because it may otherwise be a hindrance for PAIS becoming as pervasive as it otherwise would.

With respect to the development method, this means that the realization of an existing use case was regarded merely as a matter of adding a new interface. And the work with a new use case was carried out largely independently of TAIS.

So the influence of TAIS on the designed system is rather small because it for practical purposes is being encapsulated rather than developed iteratively.

2.3 The development case

The development case consists in making TAIS pervasive. The new system will be referred to as PAIS (Pervasive Administrative Information System). The results of applying UP once more is too spacious to be included here in their entirety, and is hence the complete development report has been placed placed in appendix E. Most of the descriptions in this section are taken from this report.

The first question to answer when undertaking the extension of TAIS to become pervasive, is wherein the additions or alterations should consist ? In other words, why make it pervasive ?
To answer this question, observe that PAIS is an information technology system. “An information technology system is an organized collection of hardware, software, data and communications technology designed to support aspects of some information system. An information technology system has data as input, manipulates such data as a process and outputs manipulated data for interpretation within some human activity system.” [3, p. 5]. Thus the tasks of PAIS are:

- **Storage, retrieval and organization of information needed by the employees of the Maersk Institute.** Also, capturing information becomes an option because of the enhanced sensory capabilities of pervasive computing. This does not seem apparent from the above definition.

The kinds of information that is imagined to be in focus for PAIS is that which can be dealt with in a way that exploits the system being pervasive, e.g. if a note is made about something it may initially be stored such that it is associated with the location where the user made it. In TAIS arrangement of meetings were supported, e.g. those activities taking place before the actual meeting is conducted. Because PAIS is pervasive it becomes increasingly possible to provide support during the actual meeting. This is because pervasive computing through its sensors is able to sense and react to more of what happens during the meeting. (This difference in possible degree of supported has been noted by others, in fact, the “pervasive meeting room” is an often used example of application of pervasive computing, see for instance [29, 17])

- **Facilitating communication.**

Communication can be facilitated beyond what is done by instant messaging, sms, cell phones etc. The difference lies in improved utilization of the information and i/o facilities provided by the context. The point here is not that communication takes place using means such as cameras and video-capable screens. The point is that whatever is present in the context of the communicating parties will be used. Thus the difference is that the system must possess a certain smartness allowing it to recognize the optimal means of facilitating communication in any given situation. (cameras and video-capable screens might be present with both communicating parties).

- **Provide access to the functionality of TAIS, where and when appropriate**

That the system is based on TAIS has some consequences as discussed in section 2.2.1. It also provides additional possibilities in terms of functionality. For instance the calendaring functionality can be made use of. If a person goes to check his or her mailbox-shelf at the institute at least once a week, this user might be reminded to do so on Fridays. Or, when notified (by the shelf) that a document has arrived this user may inform the calendar application that the document must be picked up before leaving the institute, so that the user will be reminded at checkout, or if he or she leaves the institute.
At this point the reader is recommended to browse through the development report in appendix E to get an impression of the functionality that was worked with; the use case diagram for PAIS is shown in figure 3 page 8 in that report.

2.4 Observations, Analysis and Criteria

This section presents the concrete observations that was made during the development of PAIS. Following that is planned result of this activity, the criteria by which a modeling technique can be evaluated as well as the analysis that lead from the observations to these criteria.

2.4.1 Concrete observations about UML and UP

Obs. 1 Access to use cases through different user interfaces

Concrete Observation
It should be possible to carry out the Reserve Projector use case both through a stationary workstation and from a PDA. There does not seem to be an obvious way to specify in UML that it should be possible to perform a given use case through a variety of different user interfaces.

General Problem
In the concrete case above one might suggest having two boundary classes, one to be used at a workstation, and one for a PDA. This is not against what UP prescribes, “Each boundary class should be related to at least one actor and vice versa.” [27, p. 183]. But it is not the standard thing to do either, “Identify one central boundary class for each human actor …” [27, p. 204].

In any respect two boundary classes does not solve the problem, because a third mobile device with different display properties may emerge after deployment of the system (e.g. it should be possible to place a reservation through a cell phone or a even a voice recognition interface, if such one were to be installed at a later time). Note that the observation here is that it should be possible to specify that a given use case can be performed through many different interfaces, not how this requirement is implemented by some mechanism (this would be a matter of design and implementation). This observation is thus about requirements modeling.

Further, the property preventing expression in UML is that the requirement is about interfaces in general, i.e. not specific interfaces constructed along with the system, but whichever interfaces find themselves available in the context of the system.

Consequences
A resulting consequence of the above issue appears in the analysis stage. It is the difficulty in stating when objects of a class are created. As is evident from
the example in [27, p. 208], part of carrying out the activity “analyze a class” is to specify when the class is instantiated. In the concrete case about reserving a projector, the example from [27] could be paraphrased “OfficialUI objects are created during the Reserve Projector use case.”, but it is not clear whether this would be correct for a voice recognition, braille or other non-gui interface. This is because interfaces that are added after deployment of the system cannot be assumed under the control of the system. Specifically their instantiation may not be the responsibility of the system, or there exist no hook through which the system can perform instantiation.

The point of why this is brought up in the first place is that what is stated at this point in the analysis model determines how the system will actually work since an analysis model is of the internals of the system. This is as opposed to the requirements model that just specifies the system from an external viewpoint. So if everything is done as usual, then the issue (of access to functionality through later introduced interfaces) is not addressed. As per the discussion of pervasive computing on page 7 it should be.

The requirement that the system should be able to adapt to user interfaces that are added after deployment thus seems to act like a force towards externalization of the user interface. This suggests that the interface perhaps ought to be modeled as an actor rather than as a class.

**Obs. 2 Internal class vs. external actor**

**Concrete Observation**

In a pervasive system, a projector is a tangible object, and therefore has an embedded computer on which some software runs. This software was named **Projector-Agent** in PAIS. The observed problem is whether the **ProjectorAgent** is internal or external to the system, that is, if it should be modeled as a class or an actor.

**General Problem**

In general, this problem occurs for every tangible object, since they on the one hand should be accessible and interact with the other objects in the system, but on the other hand may be come and go like a human actor. So regarding whether to model tangible objects using classes or actors, both cases can be argued. However in the general case it is also worth considering whether the problem is in fact to make this choice or whether it is that such a choice is required. By this is meant that maybe neither of the concepts class and actor fit the purpose.

**Consequences**

If the choice between modeling with classes or actors should not be made because a tangible object is not really either internal or external, there is the troubling consequence that the system is not sharply defined. This is highly problematic considering the definition of a system as something whose parts are either part of or not
part of the system, but never in between.

This problem is not unexpected since pervasive computing is convergent. What is converged towards might just be a whole that includes but is not limited to software and hardware. And this whole may more suitably be considered a system, of which software and hardware is merely one out of several aspects (the same way that architecture is only one aspect of a software system under development; one view of the system).

**Obs. 3  Physical aspects**

**Concrete Observation**

A concrete example encountered was in the Apply Projector use case where part of the precondition was that “the projector must be co-located with the initiating actor”. This is not very precise, e.g. it does not define how far they can be from each other.

Another problem encountered can be described in terms of the precondition of the Conduct Board Meeting use case. The precondition for Conduct Board Meeting is that the meeting has been scheduled and that the actor, an Official, who initiates the use case must be physically located where the meeting will take place. What was written in the use case was exactly this, and the problem is not with the description of the use case in the requirements workflow. Rather, it is in analysis where there within the system is no embodiment of the location with which this constraint is concerned.

**General Problem**

In general the problem in relation to the requirements model is that there are no way to define particular points of reference in the surroundings, and thus they cannot be easily referred in the specification of the system. In other words the systems presence and behaviour in the physical surroundings cannot be specified precisely.

**Consequences**

In relation to analysis, design and implementation the problem is that when reasoning about the internals of the system (in analysis and also design and implementation) there are no means by which to treat how the system relates to its physical environment. So when the events in the physical world that the system should react to are not specified with enough precision. The observed tendency in the development of PAIS were that these requirements were not addressed in analysis.

**Obs. 4  Replacement for Business Model in identification of use cases**

**Concrete Observation**

The concrete observation is the non-obviousness of how business use cases should be identified. Moreover, it was difficult to find the use cases for the pervasive
system because the business model did not provide the same support for this as in the development project in the first activity (chapter 1).

GENERAL PROBLEM

The observation here is about the difficulty of recognizing what a pervasive system should be used for and how is expressed in the business use case model and the use case model for the system. Normally identification of use cases is done from the business use case model. “A use case is suggested for every role of each worker who participates in a business use case realization and who will use the information system.” [27, p. 148].

For traditional systems, “…the business use case model presents a system (here, the business) from the usage perspective and outlines how it provides value to its users (here its customers and partners)…” For the pervasive system build for the Maersk Institute, the “business” is in a sense everything that takes place there. This is because the business use cases are characterized as something that provides value to its actors. All of our activities provide value to us in some manner, so what should be included? For traditional systems this is not a problem, because the space of what could be supported by a software system is smaller. But for a pervasive system, the space of activities that can be supported, automated, or otherwise enhanced are much larger because of the system pervading the physical world.

For instance, the simple act of turning on the light may be seen as a business use case (It provides value to he or she who does it, because this person wants the light to be turned on). The corresponding use case of the system could then be to turn on the light using a remote control (this device is already needed for the tv, so it might as well be exploited to ease the task of turning on the light or a particular lamp).

CONSEQUENCES

The unclarity of how to determine what the system should do seems to increase the risk of not building the right system.

Obs. 5 Effects of internal class vs external actor

The two observations here concern a difference in the descriptions of use cases and their realizations compared to more traditional systems. The difference is in degree rather than nature. The following will attempt to explain what it is that seems to be present to a higher degree. Both observations seems to pertain to the previous observation about the difficulty in choosing whether a given entity should be modeled as an external actor or an internal class. That this choice is not obvious can be understood such that tango-classes in a pervasive system are rather like actors.

A concrete place the first observation was made is in the Apply Projector use
case. Here the presence of a projector must be ensured since it is to be applied in the use case. This can be, and indeed was, specified in the precondition of the use case. The point here is that there is no saying whether someone unplugs or turns off the projector after the precondition has been checked. So the first observation is that describing the use cases was experienced as more difficult because many steps depend upon what events take place in the physical world, i.e. largely outside the control of PAIS. It might be said that the physical nature of some classes, the tangible objects, (e.g. a projector) leads to greater uncertainties in their behaviour; it makes this behaviour less predictable to the system, thus complicating the use case descriptions.

The second observation pertains to the same issue, but is about descriptions of use case realizations in the analysis model. Here, actors (represented through boundary classes) are treated differently from internal (control and entity) classes. The uncertainties occurring in the interactions with an actor are greater and more numerous than they are for a class under the control of the developer, as was also argued above. The observed difference is that for a pervasive system it is more difficult to design interactions with the classes whose instances are tangible objects, whether modeled as classes or actors. This is because the lesser control there is over the objects, more situations must be taken into account (e.g. the object becomes unavailable because someone has walked away with it or turned it off).

2.4.2 Observations about the general nature of the system

Relative to the properties in the table in appendix C the following can be noted about the development case.

- The qualities of Intention, History, Dynamic, Personalized, Adaption, Anticipation and Time & Space are not taken into consideration in the design of the system. This was expected to have an effect on whether the final system in fact can be said to be pervasive. Relating to PAIS the following can be said of each property:

  **Intention** is not present in the way meant by [40]. To qualify, the pervasive system would need some mechanism by which to recognize, in a given situation, what would be purposeful in relation to this intention. The intention property is thus similar to that of agents in that it specifies some degree of autonomy.

  **History** cannot be said to be a property of PAIS, because it does not record what happens to it.

  **Dynamic**, the ability for entities in the system to participate in interactions the structure of which is not fixed or predefined. This is not the case for PAIS.

  **Personalized.** PAIS and its entities does not adjust its behaviour depending on the other parties in an interaction.
Adaption is not a characteristic of PAIS and its entities because its behaviour is fixed to the predefined set of use cases.

Anticipation is not meaningful without the other properties.

Time & Space has been an issue in the design of PAIS. Primarily this is the case for time, since the functionality of PAIS depends on time and place of meetings etc.

- The concept of a habitat has not really found its way into the development, but for instance a meeting room is a habitat in the pervasive system, so it does make sense to use the concept as descriptive of the final result.

That most of these properties are not descriptive of PAIS is expected. This is because the properties are characteristic of the vision of pervasive computing. But due to the convergence of pervasive computing it is expectable that the first attempts to build pervasive systems result in systems that are in many ways alike to traditional administrative systems—we might say that these are the starting point for the convergence.

### 2.4.3 Analysis of observations and formulation of criteria

To get from the observations to some actual criteria an analysis will be performed. The observations may seem very close to criteria in themselves. This analysis will seek to uncover what the observations are symptoms of and thus what it is both in general and in specific that are the requirements of a modeling technique for pervasive systems.

The observations 2, 3 and 5 are symptomatic of the absence of means in UML to model the physical aspects of a pervasive system. Observation 2 states the problem of deciding whether to model as a class or an actor a class whose instances are tangible objects. Observation 3 shows that within the use case and analysis models there are no frame of reference against which physical constraints can be expressed or enforced. Observation 5 explains some observed consequences of the physical nature of the classes that was foregrounded in observation two, but it does not disclose a problem with UML. Based on this the following criteria seems reasonable:

**Criterion 2.1**

*It should be possible to designate what classes of a model have a physical dimension, possibly by a dedicated classifier* (A classifier is a term used in the description of UML, see [55, p. 42]. Examples of classifiers in UML are classes, interfaces, actors, use cases etc.).

**Criterion 2.2**

*In the requirements and analysis models there should be a frame of reference for reasoning about the physical aspects of the system. This is both in relation to expressing constraints (in the requirements model) and how they are enforced (in the analysis model).*
Observation 4 discloses a difficulty in deciding exactly what the pervasive system should do. The difficulty lies in identification of the right business use cases. This observation can be seen as concerned mainly with the process, but questions of process and modeling are often difficult to separate entirely. If a different activity is needed in the process in order to explain the parts of a systems environment that are relevant in the identification of use cases, then it might be that a different model is also required in order to capture these descriptions. Thus the following criterion (which is hence concerned both with the modeling technique and the process in which it is used):

**Criterion 2.3**

*There should be a suitable model on which to base the identification of use cases, thus playing the same role in development of a pervasive system as the business model does in a traditional system. Further, a clearly defined approach to obtain this model should be given. Needless to say that the approach and the model should match each other.*

Finally, observation 1 shows that there within UML is no suitable way to represent the requirement that a use case should be accessible through multiple interfaces. So the observation is about a requirement that cannot be represented. It should then be considered whether any representation can be independent of how this requirement is actually implemented. For the requirements model this seems to be the case, since it in nature represents an external view of the system. The matter is more intricate in the analysis model, because this model represents an internal view of the system. The analysis model is at a higher level of abstraction than the design model, “...it is important to note here that the analysis model makes abstractions and avoids solving some problems and handling some requirements that we think are better postponed to design and implementation ...” [27, p. 176]. So the question becomes whether the actual mechanism used to implement the multiplicity of possible interfaces is better postponed to design. This discussion will be taken up again in the activity where a modeling technique is created. For now the criterion will be this:

**Criterion 2.4**

*There should be a way to show that a given use case can be accessed through different user interfaces, some of which may not be known at the time of deployment of the system. Moreover, in analysis, there should be some means by which to show that the realization of such a use case must adhere to this requirement, without necessarily revealing the mechanism by which this is done.*

### 2.5 Conclusion

The following has been achieved in this activity:

- The foremost purpose was to obtain some criteria for modeling technique for pervasive systems. These criteria are given in section 2.4.3.
To obtain these criteria, a practical development was undertaken, and the observations made during this activity has provided a base for the above mentioned criteria. Further, the practical work gave an impression of the “flavour”\textsuperscript{1} of development of pervasive systems.

\textsuperscript{1}That is, flavour in the sense “An undefinable characteristic quality instinctively apprehended.” [46].
Chapter 3

Organizational structure and software qualities

The chapter is organized as follows. Its purpose is described in section 3.1. The method, primarily analytical, used to achieve this purpose is explained in section 3.2. Section 3.3 contains a brief comparison of organizations of people and organization of software. Section 3.4 describes how existing work on the subject of organizations and systems relates to that performed here. In section 3.5 each of Mintzberg’s five stereotypical organizations are analyzed to see what qualities they require in an information system. Section 3.6 sums up, reflects and concludes the work of this activity.

3.1 Purpose

The activity described in this chapter is about investigating the relationship between a system and an organization in which it is used. The aim is for the result to be usable in shaping a system based on how its users are organized. Its formal goal as stated in the introduction is to answer

- What is a suitable relationship between a system and its affiliated organizations and how can knowledge about the structure of these be used to select design philosophies that will help achieve such a suitable relationship?

3.2 Method

Activities I and II have been about systems for the Maersk Institute. This fact is used only indirectly in this activity since its contribution lies in the impression that these activities have given of the flavour of an information system for the kind of organization that the Maersk Institute is.

In section C.ii it was explained how the some of the forces that shape an organization are also important in relation to creation of software. This was based partly
on the similarity of software and organizations. This similarity will be reviewed to begin with.

That some forces are believed important to both the design of organizations and of software provides us with a strategy for investigating how software and organizations should relate. This is to analyze what forces shape an organization and come up with a way of accommodating these forces in the software design.

Mintzberg [43] has analyzed what forces impact the structure of organizations, and based on this proposed five stereotypical organizations.

To see how this can be exploited consider figure 3.1. It shows some intermediate steps to creating a model of some phenomenon in the real world. The phenomenon in the real world, here called the referent system [34]\(^1\) is perceived (1), and this perceived phenomenon is a mental model. A model of the phenomenon can then be created (2).

Making sense of the organizations in the real world is a difficult task, and the attention devoted to it has resulted in a large body of organizational theory. Part of this are Mintzberg’s stereotypes[43], and this can be used to build this study upon. The key point here is that by relying on Mintzberg’s stereotypical organizations, it is not necessary to carry out steps 1 and 2 to reach a model of some organization in a case study. Hereby the pitfalls of getting an imprecise or even wrong model of some real organization are avoided. One might say that the problem of giving a precise picture of an organization is taken care of by he who proposes the stereotypes.

So the method is to analyze each stereotype and determine how the forces that shape them should be used in the design of a system. Consider the following example

**Example 3.1**

*In the Professional Bureaucracy that works by standardization of skills, people en-

\(^1\)In [34], it should be noted, Nygaard et al. use the name referent system about a real or imaginary system that is being modeled; in this activity the organizations of interest are real rather than imaginary.
joy a great deal of liberty in planning and executing their jobs. This independence is exemplified in a university: “Professors . . . simply do not wish to be dependent on each other.” For this reason it would be bad judgment to make an application for time management in such a way that its users (if professors) were dependent on each other once they started using it.

The above exemplifies how one characteristic of the organization is working as a force that should be taken into account in shaping the system. It provides a concrete way to increase the acceptability of the system: just minimize dependencies among its users.

Now this example was based on a characteristic of one of the organizational stereotypes. If all characteristics of a stereotype is treated in the same way, a good picture will be given of how a stereotypical organization influences a system. Since many real organizations fits more or less into one of the stereotypical categories, such pictures will be useful in the general case of development.

### 3.3 Comparison between software systems and organizations

This section provides examples of how organizations and software systems are alike. They have the following similarities:

- They are both made of units (people/objects) that can be assembled into bigger, composite units (groups/packages) recursively. These units can be associated with each other.

- The structures in them are the result of a large variety of forces resulting from their purpose and the context they exist in. These structures form the basis for processes that the units of the system (people or objects/agents) participate in so as to fulfill its purpose.

- Coordination Mechanisms. Note the similarity between the coordination mechanism direct supervision and standardization of work processes [43] and the general ideas in the Wizard[68] and Forms [9] patterns.

On the other hand, there are also a number of differences:

- People and objects are different. This point seems obvious, but it is less true for Multi Agent Systems, the methodology of which is fastly gaining popularity.

- People have only a very limited capacity for interactions, whereas that of an object is only bounded by the capacity of networks and its host node. (Law of Demeter [?]?)

---

[43] p. 209
There are many more senses in which people can be associated with each other than it is the case for objects.

Software is always build for a reason, which is to somehow create value for its users. The reason for the existence of an organization need not be for the benefit of something or -one external to it. But both needs to participate in interactions with their context.

Obviously there are more differences, but only the above are included because the similarities are what is most interesting in this context.

Having comparatively sketched the characteristics of each, we might ask how they should relate when both are part of a bigger whole, an information system.

Their differences are why both are needed: some tasks would be practically impossible to do without a computer, while a lot of other, more complicated tasks has to be done by a person or group of persons. Their similarities makes it possible to talk of them being parallel in some respects, e.g. structure. In fact, this parallelism is an example of a way in which they might relate. Other similarities include how they are treated, probably a result of their more primary similarities. For instance, both have the structure depicted in diagrams, the processes in diagrams and descriptions etc. And both uses patterns.

To further specify the topic of this activity, define the following two spaces:

The **Informational** space is where the software exists, so the software is the part of the information system residing here. The **Organizational** space is the parts of the information system pertaining to the organization.

As mentioned before, patterns are used as a tool within both of these spaces. The patterns that would be of interest here, are those that stretches through both spaces. In other words, the set of forces that the patterns describe must have some forces in both the informational and the organizational space. The context of the pattern, and the problem it is a solution to, must exist in an information system, but neither in the organization or the software alone.

### 3.4 Existing work on system vs organization

#### 3.4.1 Global Analysis

This section will briefly explain what global analysis [23, 49] is and why the results aimed at in activity III address an issue different from the one addressed by global analysis.

Global Analysis is an activity performed by architects to analyze the factors that influence the architecture and to develop strategies for it to accommodate them. The analysis starts out with identification of the factors which fall in three categories, organizational, technological and product factors.

Organizational factors are facts about the organization of the developers that can influence the architecture. These include the budget, development schedule
3.5 Analysis of each stereotypical configuration

In the following, each of Mintzberg’s five stereotypical organizations will be considered from the perspective of how it affects a system.

Leifer [35] has proposed, for each stereotype, what sort of system fits its structure. But the systems are characterized mainly by how centralized they are, using four categories ranging from pure mainframe-terminal to distributed systems with no central servers. This is of little help as input to the software development process (additionally, the article is from 1988—a lot has happen since then). A similar study has been performed by Jordan [30] who investigated the relationship between an organization’s structure and an information system that supports its operations. It proposes a number of hypothetical matches between each of Mintzberg’s stereotypes and generic information systems, based on an analysis of the information flow in the different organization types. The hypotheses are mostly validated by a study of a large bank containing as its divisions several instances of each stereotype.

The analysis is performed by asking a number of questions for each stereotype. What requirements are imposed on the system by the characteristics of each organizational stereotype? What functions in the organization can be aided by a...
3.5.1 The Simple Structure

<table>
<thead>
<tr>
<th>Prime Coordinating Mechanism:</th>
<th>Direct Supervision</th>
</tr>
</thead>
<tbody>
<tr>
<td>Key Part of Organization:</td>
<td>Strategic Apex</td>
</tr>
<tr>
<td>Main Design Parameters:</td>
<td>Centralization, organic structure</td>
</tr>
<tr>
<td>Situational Factors:</td>
<td>Young, small; non-sophisticated technical system; simple, dynamic environment; possible extreme hostility or strong power needs of top manager; not fashionable</td>
</tr>
</tbody>
</table>

The simple structure is characteristic of such organizations as entrepreneurial firms, a government headed by an autocratic politician, middle-sized retail store [42].

Thus nearly all power in the organization resides with the CEO. Both of [35, 30] argues that the often small size of organizations with the simple structure implies little need for applications beyond stand-alone systems for e.g. bookkeeping and word processing. The study [30] includes two of these organizations, and only partly validates this view. So what little can be said is that if a system is designed particularly for such an organization, it should be done using the tool philosophy in B. vi page 6

3.5.2 The Machine Bureaucracy

<table>
<thead>
<tr>
<th>Prime Coordinating Mechanism:</th>
<th>Standardization of work processes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Key Part of Organization:</td>
<td>Technostructure</td>
</tr>
<tr>
<td>Main Design Parameters:</td>
<td>Behavior formalization, vertical and horizontal job specialization, usually functional grouping, large operating-unit size, vertical centralization and limited horizontal decentralization, action planning</td>
</tr>
<tr>
<td>Situational Factors:</td>
<td>Old, Large; regulating non automated technical system; simple, stable environment; external control; not fashionable</td>
</tr>
</tbody>
</table>

That the prime coordination mechanism is standardization of work processes, combined with a high degree of horizontal specialization, enables parts of these to be automated. As Mintzberg notes, “Work processes are standardized when the contents of the work are specified, or programmed”([43] p. 5). Though by
programming is not meant the kind done on a computer, the extreme example of what meticulous specification can enable is that some jobs in automobile factories have been taken over entirely by robots. Automation comes in various degrees e.g. an application can help in monitoring and controlling the progress of welding together the elements of a container ship (Some of the welding is even done by robots, see [?]), but the operating core still comprises a number of humans to the rest of the tasks. Depending on degree, this can push the organization towards the shape of an administrative adhocracy; if the operating core becomes wholly automated it is effectively truncated “so that the administrative component that remains can be structured as an adhocracy” [43, p. 259].

Before proceeding, it should be noted that increasing the automation in an organization should be done with care. [63] gives an example of how automation in a concrete case reduced the need and possibility for direct conversational communication between colleagues, which from a social perspective had negative consequences.3

The high degree of formalization found in machine bureaucracies causes a lot of paperwork, which is an obvious target for computerization, according to [35]. Systems that help manage paperwork, in a sense automates it, need not be integrated, because the task is primarily to enforce the formalization already existing, according to [30].

The elaborate work with standardizing work processes is carried out by the technostructure. In machine bureaucracies this part of the organization thus becomes dominating[43]. So software used in a machine bureaucracy must suit the needs of the technostructure, that is, whatever kind of changes its work results in should be easily implementable in the system. The work that should be supported is specification of work processes, and in general, conformance with the rules of the business; the business rules.

Business rules as known within software development seems to provide the kind of conceptualization required to partly obtain flexibility in support for the work of the technostructure. A business rule is defined as

A business rule is a statement that defines or constrains some aspect of the business. It is intended to assert business structure or to control or influence the behavior of the business. The business rules which concern the project are atomic <- that is, they cannot be broken down further. [21, p. 4]

Business rules is a difficult subject matter to capture formally, because their definition is so wide. Therefore the use of them will in itself not suffice to ensure adequate flexibility of the system. Obtaining the correct scope requires a careful analysis of what kinds of rules are likely to change, and what kinds it is safe to hard-code into the system. Fortunately this task is eased due to the environment being simple and stable, because stability implies, if not that it is static, then at

---

3The example is focused on automation, though it is of police-work that is not really performed in a machine bureaucracy. The qualitative effects described is, however, largely independent of the concrete organizational setting, and thus interesting for what it tells about automation.
least that it is predictable [43]. A possible way of achieving flexible applications is through use of an adaptive object model (AOM), [73]. "An Adaptive Object Model is a system that represents classes, attributes, and relationships as metadata. The system is a model based on instances rather than classes. Users change the metadata (object model) to reflect changes in the domain. These changes modify the system’s behavior." [73, p. 50]. This facilitates building a system that is in effect what Malone [37] designates radically tailorable. Such systems “allow end users to create a wide range of different applications by progressively modifying a working system” [37].

To sum up, the technostructure is the dominating part in a machine bureaucracy. Its work lies in standardization of processes which is the prime coordination mechanism in these organizations. The paperwork and meticulously specified workflows that is found here is an obvious target for computerization. That the technostructure should have the ability to perform its work after this computerization has taken place implies a need for flexibility in these systems. The design philosophy of radically tailorable systems allow such flexibility to be created. It can be done possibly through use of an adaptive object model.

### 3.5.3 The Professional Bureaucracy

Mintzberg’s summary of the stereotype is repeated in the following table.

| Prime Coordinating Mechanism: | Standardization of skills |
| Key Part of Organization: | Operating core |
| Main Design Parameters: | Training, horizontal job specialization vertical and horizontal decentralization |
| Situational Factors: | Complex, stable environment; non-regulating, non-sophisticated technical system; fashionable |

The prime coordination mechanism is standardization of skills. This is due to the highly complex nature of work done by doctors, researchers, teachers and other professionals employed in these organizations. So these people fall within the category of knowledge workers, explained by “…the knowledge worker who puts to work what he has learned in systematic education, that is, concepts, ideas and theories, rather than the man who puts to work manual skill or muscle.” [14]. This explains what is indicated by the name of the predominant standardization method in professional bureaucracies, that the skills rather than the actions are standardized. But standardization of processes was identified above to enable automation, sometimes even replacement with robots, so knowledge workers must be supported differently. Kidd [32] has studied how knowledge workers work, and argues that their work is beneficial through the way it changes them, rather than
through the data they produce. They are informed in the sense “To give form to the mind, . . ., to furnish with knowledge”\(^3\). Based on this she derives an array of guidelines for systems supporting knowledge workers. The first captures the main conclusion, namely that it should be avoided to try to “help knowledge workers in ways which involve the tool in understanding the information it is holding or predicting what the user wants to do with it”\(^{[32]}\).

That these kinds of organizations are highly decentralized means a great deal of power resides with the operating core, those who do the actual work. As Mintzberg notes,

> The professional’s power derives from the fact that not only is his work too complex to be supervised by managers or standardized by analysts, but also his services are typically in great demand. This gives the professional mobility, which enables him to insist on considerable autonomy in his work. When the professional does not get the autonomy he feels he requires, he is tempted to pick up his kit bag of skills and move on. \(^{[43, p. 195]}\)

This means that if a system is to support the work of the operating core in professional bureaucracy it is vital that these people accept the system and feels it is of benefit to them rather than just the organization as a whole. Thus, the acceptability especially important for systems made for the operating core of a professional bureaucracy. Acceptability is the degree to which users accept the system. A range of issues may affect this quality, e.g the usability of a system. Another obvious way to increase acceptability is to refrain from having the system take away any privileges and power from its users. For instance, knowledge workers have the privilege to plan their own time, so a calendar application should not allow managers to “book a meeting” rather than arranging it in the traditional conceptual way. Also, the guideline advised by Kidd will increase acceptability by avoiding that the users think of the tool as being stupid or constraining. A study by Davis \(^{[11]}\) (and supported by \(^{[1]}\)) shows a high correlation between perceived usefulness, defined as “the degree to which a person believes that using a particular system would enhance his or her job performance”, and self predicted current and future usage.

To sum up, the work performed by the operating core in professional bureaucracies is complex, thus making the prime coordination mechanism standardization of skills. The great deal of power located at the operating core in turn leads to the prioritization of acceptability for systems used by these knowledge workers, since they have the power to reject using it. The complex nature of the work performed in the operating core means that any tools that support these knowledge workers, should not do so in a way requiring the tool to understand the information it deals with. An additional way to increase acceptability is by maximizing perceived usefulness and usability.

\(^3\)\textit{Oxford English Dictionary} \(^{[46]}\)
3.5.4 The Divisionalized Form

<table>
<thead>
<tr>
<th>Prime Coordinating Mechanism:</th>
<th>Standardization of outputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Key Part of Organization:</td>
<td>Middle Line</td>
</tr>
<tr>
<td>Main Design Parameter:</td>
<td>Market grouping, performance control system, limited vertical decentralization</td>
</tr>
<tr>
<td>Situational Factors:</td>
<td>Diversified markets (particularly products or services); old, large; power needs of middle managers; fashionable</td>
</tr>
</tbody>
</table>

A study by Jordan [30] of the organization Corporate Banking Hong Kong supports the view that “The generic information system for the divisionalised form will be a series of distinct computer systems in each division and often in departments within the individual organization, forming parts of a large MIS. The principle feature the systems have in common will be the output required by headquarters, which needs to monitor and control the performance of the division.” ([30] p. 11). So in a sense the information system will be parallel to the organization, i.e. made of subsystems with standardized outputs.

The company 3M, selling more than 50,000 products in 200 countries, is a classical example of divisionalized organization. Sawhney [58] explains how the fragmentation of this organizations information system was disadvantageous with respect to customer relations. For instance, “…it was impossible for the company to answer a simple but crucial question: How much business do we do with a specific customer?” [58]. The fragmentation also had a negative effect when the company moved onto the Internet:

Decentralized and poorly coordinated, 3M’s Web sites mirrored the structure of the company itself, with customers having to visit several different sites to get information on related products. A health care professional, for instance, might buy products and services from 3M’s pharmaceutical division, skin health division, medical-surgical division, medical specialties division, and office supplies division. To get information about these products, she would have to register separately with each of these divisions’ Web sites, receiving a different password and user ID, and she would have to familiarize herself with the sites’ different designs and navigation schemes. When she bought something from one unit, moreover, no record of that transaction would be available to any other unit, which made it impossible for her to receive fast, consistent customer service from the company. In short, 3M was out of sync. It presented many different and often contradictory faces to its markets, all of them reflecting its internal silos rather than its customers’ needs.

3M solved these coordination problems by building a global data warehouse. Given that the data held by such a warehouse often is there before a warehouse is created (as with 3M), one may consider the option of a distributed database. “In a distributed database system data is physically stored across several sites, and each
A site is typically managed by a DBMS that is capable of running independently of the other sites.” [53, p. 597]. Whether this is an option depends on the performance requirements, since “Distributed data management, in the final analysis, comes at a significant cost in terms of performance, software complexity, and administration difficulty. This observation is especially true of heterogeneous systems.” [53, p. 608]. The problem of differing formats of the data held by the divisions has been devoted some effort, for instance [62] describes a framework for automatic schema translation. The Object Management Group has put forth the Common Warehouse Metamodel Specification [67]. “The main purpose of CWM is to enable easy interchange of warehouse and business intelligence metadata between warehouse tools, warehouse platforms and warehouse metadata repositories in distributed heterogeneous environments.” [67, p. xvi].

So what can be derived from the literature is that divisionalized organizations can harvest a significant advantage by using information technology to present a uniform and consistent interface not only to its environment but also to within the organization. Keeping in mind the purpose of this analysis, it seems that software for such organizations are subject to a force in the direction of centralization. But this is centralization from the perspective of the consumers of the information managed by the software. The producers of the information are the divisions. Since their output is already standardized (this is the prime coordination mechanism in the divisionalized form), the step to standardize the product descriptions entered into a database is not unreasonable, and may only be required to a limited degree due to ongoing advances in the area of metadata interchange and translation.

### 3.5.5 The Adhocracy

<table>
<thead>
<tr>
<th>Prime Coordinating Mechanism:</th>
<th>Mutual adjustment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Key Part of Organization:</td>
<td>Support staff (in the Administrative Adhocracy; together with the operating core in the Operating Adhocracy)</td>
</tr>
<tr>
<td>Main Design Parameter:</td>
<td>Liaison devices, organic structure, selective decentralization, horizontal job specialization, training, functional and market grouping concurrently</td>
</tr>
<tr>
<td>Situational Factors:</td>
<td>Complex, dynamic (sometimes disparate) environment; young (especially Operating Adhocracy); sophisticated and often automated technical system (in the Administrative Adhocracy); fashionable</td>
</tr>
</tbody>
</table>

The adhocracy comes in two forms, operational and administrative. The name *adhocracy* means that immediate purposes shape the organization [66]. The operational adhocracy innovates and solves problems directly on behalf of its clients. A
characteristic of it is that its administrative and operating work tend to blend into a single effort [43]. The administrative adhocracy performs projects to serve itself, and the operating core is truncated, by having its work outsourced or automated [43].

Travica [66] has studied the relationship between the use of information technology and non-traditional organizational characteristics. His study shows that “IT is related inversely to ... centralization and formalization.” [66, p. 1241] and “The finding ... suggests that ICT [Information Communication Technology] enables the non-traditional organization to a significant extent.”[65, p. 183].

Olson and Bly [48] performed a case study of work in an adhocracy (distributed into two units). They suggest three goals in development of tools for supporting this work, two of them concerned with reciprocal interdependency, which is the situation “where each of two facilities also has resources required by the other, and coordination of the two is required.” [48, p. 221]. The goals are:

- “To make two separate physical environments more like a single environment.” This is a goal of e.g. video conferencing.

- “To improve the efficiency of reciprocal interdependence.” An example of this is to substitute email for telephone conversations because it eliminates unnecessary social conversation [48].

- “To increase the capacity of reciprocal interdependence by improving accessibility to shared information.” An example of this is to replace particular use of email with CVS.

So to sum up, the way work is performed in an adhocracy means that communication is important, in fact, information communication technology helps enable the kind of organization that the adhocracy is. Goals in a system for an adhocracy are to improve sharing of information, and if the parties that communicates are separated physically, to make the two physical environments more like a single environment.

### 3.6 Conclusion

The purpose of this activity were to propose a design philosophy for each stereotype. This has been done, and the results of the analysis is shown in the following table:
3.6 Conclusion

<table>
<thead>
<tr>
<th>Stereotype</th>
<th>Qualities in SW</th>
<th>How to achieve them</th>
</tr>
</thead>
<tbody>
<tr>
<td>Simple Form</td>
<td>tool based information system</td>
<td>Use a collection of different programs, or in a custom made program use the tool philosophy[38].</td>
</tr>
<tr>
<td>Machine Bureaucracy</td>
<td>Flexibility for the work of the technostructure</td>
<td>use a design philosophy like radical tailorability[37], perhaps together with an adaptive object model[73].</td>
</tr>
<tr>
<td>Professional Bureaucracy</td>
<td>increase acceptability by...</td>
<td>avoiding to make the system such that it needs to understand the data it holds or the work it supports.</td>
</tr>
<tr>
<td>Divisionalized Form</td>
<td>present uniform data to other parts of organiza-</td>
<td>centralize the interface to data through data warehousing, possibly managing differences in output formats of organizational units through schema translation methods.</td>
</tr>
<tr>
<td>Adhocracy</td>
<td>enable communication</td>
<td>improve sharing of information, see 3.5.5</td>
</tr>
</tbody>
</table>

3.6.1 Discussion

The subject in this activity was how a software system should relate to the organization in which it is used. This is a rather broad subject, since there are many senses of the word “relate” that can be assumed, as was indicated by the four examples in the introduction, page 18. The findings in this chapter are mainly about the third kind, where particular qualities of the organization act as forces towards having the software system exhibit particular qualities (e.g. tailorable or support for communication).

The analysis performed was centered around Mintzberg’s stereotypical organizations. The nature of the work and how it shapes the structure of the organizations provides the core of the his argumentation for each stereotype, so the analysis in this chapter naturally reflects this. Apart from the structure, it could be interesting to consider how other aspects of the organization influence software.
Chapter 4

A modeling technique for pervasive systems

This chapter will explain the modeling technique for pervasive systems that has been found from the criteria presented in chapter 2. The chapter is structured as follows.

Section 4.1 explains the general approach used to come up with the semantic concepts and notation which constitutes the core of the modeling techniques proposed.

Section 4.2 presents some parts of what can be denoted the philosophy of modeling. This starts with a discussion of what models are. Next follows a presentation of some models of reality, or world-views, that are useful in relation to software. The section closes with a discussion of how models relate to software, based primarily on the design principles behind UML.

Section 4.3 presents the modeling technique that is the result of activity IV. Section 4.4 concludes and sums up this chapter.

4.1 Method

The approach that will be used to obtain a modeling technique we suggest explained by comparing the development of this modeling technique with the development of a piece of software. This comparison is chosen as a mean to explain what has been done without implying any other similarity between creation of software and models. For development in general there are three major parts.

1. Some concrete requirements to the final result are needed. In software these are captured as use cases. For the modeling technique, these requirements are the criteria of section 2.4.3.

2. What is also needed to obtain the final result (be it software or a modeling technique) is some knowledge about the class that this result belongs to, that
A MODELING TECHNIQUE FOR PERVERSIVE SYSTEMS

is, its form, structure, representation, intended use, and the means by which it can be handled.

In the case of software, this includes modeling languages and a development process. What will be used here, where the result is a modeling technique, is the knowledge that exists about models. This knowledge is less formalized than for software; there is no formal process for obtaining a modeling technique based on some requirements to it. But there are various theories about what models are, and there is a set of concepts used to treat them as a subject.

3. The last component of a software development project is to perform a test to see if the result matches the requirements. When the result is a modeling technique, this is done similarly, by putting the modeling technique to use in the concrete cases foregrounded by the observations.

4.2 The Philosophy of Modeling

This section will review and present the existing body of theory about models to the degree it is relevant for the work actually done. The presentation will fall in three parts.

Firstly, subsection 4.2.1, contains a discussion of what a model is. Secondly, subsection ?? presents a number of world views, or models of reality. A model is always a model of something, so let this something be denoted a referent system. The ways that humans perceive and understand this referent system forms the basis for what models are relevant and should be used. The criterion 2.3 stated that there should be a suitable model on which to base the identification of use cases for a pervasive system thus playing the same role that the business model does in the unified process. For such a model the referent system that should be understood is the “business” or problem domain that the pervasive system should help administer. This business is part of our world, so it will be investigated what “world views” described within the discipline of philosophy that are most convenient to employ when wanting to create a model of this business.

Thirdly, subsection 4.2.3 the Unified Modeling Language has been a very successful language for creating models of software, so the design principles underlying it will be explained and taken into consideration.

4.2.1 What is a model?

Models are used in various places as a tool of contemplation, so it is only expectable that the definitions span wide, indeed [69] has termed this the Model Mudlcle. Whether \( A \) is a model of \( B \) depends on the perspective, in fact as noted by Wartofsky:

To propose something as a model of (an) \( x \) is to suggest it as a way of representing \( x \) which provides at least some approximation of the actual situation;
moreover, it is to admit the possibility of alternative representations useful
for different purposes. [69, p. 2]

And the actual situation is always filtered by the act of perception such that what
may be a good model in one person’s mind is not in another’s. To name a few
types of models, there are mathematical models, physical scale-models of build-
ings, models in sociology. And then there are models used to build software. Rum-
baugh [55] gives the following, similar, definition of a model:

A model is a representation in a certain medium of something in the same
or another medium. The model captures the important aspects of the thing
being modeled from a certain point of view and simplifies or omits the rest.

What is given here is only an extremely abbreviated presentation of models. Refer
to [69, 15] for a more thorough discussion.

4.2.2 Models of reality

In previous chapters the process of modeling was explained in terms of referent and
model systems, a view obtained from Nygaard [34]. In [34] the referent system is
either a real or imaginary system. If the system is real then this view may be seen as
resting on what Winograd names the rationalistic tradition, summed up as follows.

(1) We are inhabitants of a ‘real world’ made up of objects bearing prop-
erties. Our actions take place in that world. (2) There are ‘objective facts’ about
that world that do not depend on the interpretation (or even presence) of any
person. (3) Perception is a process by which facts about the world are (some-
times inaccurately) registered in our thoughts and feelings. (4) Thoughts and
intentions about action can somehow cause physical (hence real-world) mo-
tion of our bodies. [72, p. 31]

This tradition has a close correlation with science [72]. But in the explanation given
by Nygaard the referent system can also be imaginary. The difference between
what is real and what is imaginary seems obvious and intuitively understandable,
but as Winograd continues,

Much of philosophy has been an attempt to understand how the mental and
physical domains are related—how our perceptions and thoughts relate to
the world toward which they are directed. Some schools have denied the
existence of one or the other. Some argue that we cannot coherently talk
about the mental domain, but must understand all behaviour in terms of the
physical world, which includes the physical structure of our bodies. Others
espouse solipsism, denying that we can establish the existence of an objective
world at all, since our own mental world is the only thing about which we
have immediate knowledge. Kant called it “a scandal of philosophy and
of human reason in general” that over the thousands of years of Western
culture, no philosopher had been able to provide a sound argument refuting
psychological idealism—to answer the question “how can I know whether
anything outside of my subjective consciousness exists?” [72, p. 31]
This quote shows that it is not exactly clear where to draw the boundary between imaginary and real systems, indeed, if there are systems in the physical world or if they only become systems by being perceived as such. The point here is that it is not clear upon which of these two world-views that traditional software engineering is build.

Heidegger disagreed with Kant’s remark in the above quote, and took that stand that the scandal was not the lack of a sound argument, but that such an argument was at all required. He proposed an entirely different worldview that seems to bear particular relevance to pervasive computing.

This world view is characterized by what he calls *thrownness*, used to describe the situation of being in the world. What is meant by this concept is the characteristic of being that one in a sense is captured in it. Any actions we take in the world have consequences. Therefore we might want to think about what actions to take. But even doing this is a decision which has consequences, so we are constantly thrown into the reality of being. In other words thrownness describes a view by which it is impossible to withdraw from the world to think about what actions to take, because even this withdrawal or state of non-action-taking qualifies, in a sense, as an action.

King explains this property of being captured in the world, “Fallenness, or “falling captive to the world” (Verfallen) is a trend toward the world which is basic to man’s being, . . . tendency to give himself away to things, to scatter himself in his occupations in company with other people . . .” [33, p. 51].

The point of this worldview is its relevance to pervasive computing. This relevance lies in the alternative explanation it provides of the act of perception. Rather than having a physical world with physical objects, these objects only come into existence when a breakdown occurs, as explained in the following example:

One simple example he [Heidegger] gives is that of a hammer being used by someone engaged in driving a nail. To the person doing the hammering, the hammer as such does not exist. It is a part of the background of *readiness-to-hand* that is taken for granted without explicit recognition or identification as a object. It is part of the hammerer’s world, but is not present any more that are the tendons of the hammerer’s arm. The hammer presents itself as a hammer only when there is some kind of breaking down or *unreadiness-to-hand*. Its ‘hammerness’ emerges if it breaks of slips from grasp or marks the wood, or if there is a nail to be driven and the hammer cannot be found. The point is a subtle one, closely related to the distinction between thrownness and reflection on one’s actions, as discussed above. As observers, we may talk about the hammer and reflect on its properties, but for the person engaged in the thrownness of unhampered hammering, it does not exist as an entity.

[72, p. 36]

Note that the quote says that the hammer does not exist for the person engaged in unhampered hammering. This is exactly one of the properties wanted in pervasive computing as described in Weiser’s [70] original article, “The most profound technologies are those that disappear”. So for pervasive computing to become a
tacit and natural part of our lives, just like the hammer is it in the above example, it should be sought to avoid breakdowns in the interactions with this technology. This observation will form the basis for part of the modeling techniques presented later in the chapter.

### 4.2.3 Models of software

#### Design Models

Most models of software are design models which are different from the models mentioned above in that what they are models of does not necessarily exist. When it is appropriate to say that something exists is a philosophical question in its own right, and not one to be answered here. Suffice to say that a software system is for the purpose of this discussion seen as being created, thus coming into existence, during the development of it. So what seems to be shared among the models used in requirements, analysis, and design, is that one of their primary purposes are to define what is to come into existence, and to an increasingly higher level of detail culminating in the implementation which, for all practical purposes, is the system.

It may hence be accurate to say that kind of models used in software are reverse, in that what they model is created from the model, rather than the model being created from that which it is a model of.

#### Unified Modeling Language

The Unified Modeling Language is a general-purpose modeling language that is used to visualize, construct, document and specify the artefacts of a software system. The name refers to the fact that the language is unified or unifies across historical methods, the development lifecycle, application domains, implementation languages and platforms, development processes (it is process independent), and internal concepts (in order for the concepts of UML to be broadly applicable).

The two main aspects of a software system that is modelled with UML are static structure and dynamic behaviour. It also allows modeling of implementation constructs, and it facilitates organization of the model itself in order to enable human creation of models that are comprehensible to humans (hierarchical models rather than large, flat models).

In relation to the creation of a modeling technique for pervasive systems it is relevant to consider the extension mechanisms of UML. These are constraints, tagged values and stereotypes. Constraints are semantic restrictions represented as text expressions in a particular language (OCL, C++, pseudocode, natural language etc.). A tagged value is a pair of strings, the first denoting a property (e.g. “author”) and the second denoting the value of this property (e.g. “Yasunari Kawabata”). A stereotype is “a new kind of model element defined within the model based on an existing kind of model element.” [55, p. 449]. Examples of stereotypes are the boundary, entity and control classes used in the analysis workflow of UP—they are defined from UML’s class element.
Refer to [55, 54] for an extensive presentation of UML and its intended usage.

4.3 Modeling Primitives

A number of modeling primitives are proposed that should fulfill the criteria from chapter 2. These criteria pertain for the most part to the concepts behind modeling, and what reasoning it should facilitate. They are presented immediately below and it will afterwards be discussed how they serve to fulfill the criteria that relate to them.

4.3.1 Tangible Object

It is proposed to use a dedicated stereotype to model a class whose instance is a tangible object, a tango-class in the following. The criterion 2.1 page 43 stated that it should be possible to designate which classes in a model have a physical dimension, that is, whose instances are tangible. This is possible to accomplish using the extension mechanisms build into UML, in this case a stereotypical class.

Concept

The underlying problem here was not just about notation but also about the fact that a class of tangible objects was observed to be different from both classes and actors. They were different from classes because it was sometimes suitable to model them as actors. This is the manifestation of an intuitive feeling that tango-classes were more external than a traditional class. Similarly, tango-classes were different from actors because it was sometimes suitable to model them as classes, which indicates that they sometimes are best seen as internal to the system, that is, part of it. This is interpreted to mean that neither of these two concepts matches the meaning of a tango-class. The exact conceptual meaning of a tango-class is simply a class of tangible objects. A definition of a tangible object is given below:

**Definition 4.1**

A TangO is an object that exists as a physical as well as an informational entity. Interactions with a TangO can be done either in the informational or the physical dimension.

An object is traditionally defined as an entity having identity, state, and behaviour. A tangible object has this both in the informational and the physical sense. It has only one identity.

That the tangible object only has one identity means that its informational and physical parts are not two objects, but rather two aspects of the same object.

The state of a TangO is defined both in terms of the values of its informational attributes and its physical state (its location, orientation, position of switches). An attribute can be informational (e.g. a logic state), physical (e.g. if location is not represented informationally) or both (e.g. the value of a switch).
The behaviour can come to expression either through traditional message passing in the informational space, or, in the physical space through sensors and actuators.

**Notation**

Having thus described the concept of a tango-class, its notation must be discussed. In UML the classifiers (classes, actors, use cases) are depicted as simple drawings which can be connected by links (associations, aggregation, generalization, etc.). The notation may depend on the purpose of the usage of the tango-class, just like a class can be depicted in different ways, with stereotype symbols and with or without showing attributes and methods.

Basically a tango class is still a class, and the UML notation for a class thus forms the basis for a notation for a tango class. In requirements capture, tango classes are shown as ordinary classes. The standard designation of a stereotype in class diagrams in analysis and design is by use of the «typename» notation, so this is suggested for tango-classes as well.

In the Locational view introduced below, it is the instances of tango-classes that are shown in the maps. Their stereotype is indicated by a red dot.

This concludes the task of describing a modeling concept which fulfills the criterion from above. In the following an example is given of application of this modeling concept.

**Example 4.1**

An example of a class of tangible objects in the development case in activity II is a Projector. The suggested notations in the requirements model and in analysis and design are shown in figure 4.1.
4.3.2 Physical Surroundings

The criteria 2.2 page 43 stated that there was a need for a frame of reference for reasoning about the physical aspects of the system, that is, a model of them. To find a way of modeling these physical aspects, an overview of the existing work in this area will be given. This work focuses on location modeling, but still suffice to serve as a bag of ideas from which a concrete modeling technique can be assembled.

It is proposed to add a view in the sense used by UML, “A view is simply a subset of UML modeling constructs that represents one aspect of a system.” [55, p.23]. This will be called the Locational View, since the name “Physical View” is already used as a name in UML [55, p.32] (it contains deployment diagrams etc.).

The selection of what to include in this view is based on an an analysis of what aspects of the physical parts of the system and its environment are relevant in requirements capture and analysis—the two development activities mentioned in the criteria.

The central diagram type for the static view in UML is a class diagram; in the view added here the central diagram is a map of a habitat.

Existing work on modeling physical aspects

The term “physical aspects of the system” used in criterion 2.2 is more general than just to include locational information. For instance the design of the physical form of a tangible object; what it will actually look like and what materials it is made of. Though there is no clear boundary of what is relevant to the software driving a pervasive system, issues such as the shape and material of individual tangos are not the primary concerns from a software point of view. The focus here correlates with that of the existing work about modeling the physical aspects of pervasive and ubiquitous systems, primarily location modeling. An overview of the existing work on location modeling is given by Domnitcheva [13], who gives examples of different kinds of models.

Physical location models are characterized by employing a global geographic coordinate system, e.g. location is given as a <longitude,latitude> pair. This is the kind of model underlying the GPS system.

Geographical location models consists of geographic objects, “such as countries, cities, and also zip codes, postal addresses and so on.” [13, p.14]. These objects have the property of a clear hierarchical organization. In the development case in activity 2, a geographical model could be scaled floor plan of the institute.

Geometric models represent locations and objects as n-tuples describing positions, areas and volumes. A geometric model for the development case in activity two could be a 3D model of a tangible object, such as a projector.

In symbolic models, location is referred to by abstract symbols, locations are sets and objects are the members of these sets. A geographical model seems to be a kind of symbolic model [13].

Another kind of location information is semantic location, as employed in the
Cooltown Project [50]. Here a location may for instance be at a bus-route or the like, which does not have a precise physical coordinate, but remains useful. In the case of the system from the development case, a semantic location of a PDA could be a jacket, a person, or a car.

So depending on the perspective there are a variety of different models that can make sense.

### Selection of modeling primitives

For the problem at hand, there are two perspectives, the one employed in requirements capture, and the one used in analysis.

The model should be a frame of reference with respect to physical aspects. The grouping of these are done in terms of habitats:

**DEFINITION 4.2**

A **habitat** is an area (see definition 4.4) that is modeled by a map. The name habitat comes from biology where it is defined as “a place where animals live”[?]. In relation to pervasive computing it will be taken to mean an area that in itself constitutes a whole which is conceptually bounded from its immediate surroundings either by physical barriers or by differences in use. For instance a room, a dining area of a large open space, a floor in a building. It is, in a sense the place where a tangible object lives.

**REQUIREMENTS CAPTURE**

In requirements capture, the goal is to specify what the system should do. This is done in collaboration with the users, so the modeling concepts should be understandable to them. According to [13], symbolic models are easier to understand for humans than geometric or physical models, so they are the obvious starting point for a model that is used by humans, as is the requirements model.

The symbols in symbolic models are things that have a known position. In this case physical objects:

**DEFINITION 4.3**

A **physical object** is a solid entity occupying a volume of space in a habitat. It has a position, may be able to contain other physical objects, or points, paths and or areas as defined below.

Physical objects are needed to give an adequate picture of the surroundings. It is easier for a user to identify a drawing as a map if the map has some sort of visual resemblance to that which it is a map of. An example is that in a map over a room, the walls, doors, tables and perhaps even chairs provide a representation of the known parts of a room. This way the location of e.g. paths and areas can be comprehended better because it can be comprehended in relation to something the location of which the user already has a mental model of.
To motivate the following modeling primitives consider the sentence “The User enters the presentation area, and commands the system to show the first slide using the projector.”. This was taken from one of the use case descriptions in the development case of activity II. In order for this to make sense, the “presentation area” should have a clear definition. This definition can be specified in absolute coordinates in a room, or it can be defined as an area of a given size and shape lying between a projector and a screen it can project to.

Thus, one of the modeling primitives is an Area, it has a position which is either absolute or relative to a set of objects. Similarly it makes sense to be able to refer to concrete Points and Paths. Each of these concepts are more formally defined below.

**Definition 4.4**

**Point** has a particular fixed or relative position on a map. It also has a name and, if relative, associations to the entities its position is defined in terms of. Further, a point can be compared to another geometrical entity to obtain a distance to this entity.

**Path** a path is a curve on a map. It has the same properties as a point. Beyond this it can be traversed by some physical object or person.

**Area** is a closed curve. It has the same properties as a point. It can contain physical objects, areas, and paths wholly or partly, and these can exit and enter the area.

The tangible objects are what the user interacts with, so to him or her they represent the embodiment of the system. The concept of a tangible object has already been defined, but in the relation to their depiction on a map it should be mentioned that (1) they have a name and (2) their property of being tangible is shown by a red dot.

Note that for requirements capture the distinction of tangible and physical objects may not be accurate. If for instance it should be possible to detect that something is lying on a table, the sensor used for this purpose may not be shown. If, for instance this sensor is build into the table, the table may still be depicted as a physical object. In other words, the objects designated tangible in the map used in requirements capture are primarily those that the user directly interact with, or expect some active behaviour from. That is, those which are perceivably tangible to the user.

**Analysis**

In the analysis the perspective is on the internals of the system, but relatively undetailed. The additional parts of the physical surroundings that should be reasoned about in analysis are the existence and initial positioning of sensors and actuators. Since analysis is independent of the implementation platform, the actual brand and other details about the sensors are not specified. This means that the position of
sensors need not be specified accurately, since it may depend on the concrete one chosen. The purpose is just to give an idea of which sensors are needed and a rough overview of their position. So the same map can be used as was used in requirements capture.

**DEFINITION 4.5**
*Sensors and Actuators are a kind of widgets (c.f. the pattern in chapter 5) and are shown as black boxes with a blue dot.*

**EXAMPLE 4.2**
*In the development case in activity II, an example of a habitat is the room wherein presentations take place. A map of this habitat is shown in figure 4.2. The presentation area designate the place in which a connection should be made between a user’s PDA and a projector, if this user is expected to perform a presentation.*

*This means that the problem encountered in the Apply Projector use case described in observation 2.4.1 can be solved. Rather than having the imprecise precondition “the projector must be co-located with the initiating actor”, we can now say that “The initiating actor must be within the presentation area”, since this area is a well defined part of the physical surroundings.*
4.3.3 Break-down centered business model

Criterion 2.3 page 44 requires a suitable model for identification of use cases, and a clearly defined approach for creating this model. In traditional software development this model is a business model.

The problem with a traditional business model

The business model presents a system (the business) from a usage perspective, depicting how it provides value to its users. Implicitly, when the business model is used in software development, the business model describes what the system should be aiding with.

When the move is made from traditional to pervasive systems the reach of the system increases, i.e. there is a much larger set of activities that can be assisted by pervasive technology than by traditional desktop applications.

This means that there is a problem of selecting what to include in a business model, because the traditional “everything that can be supported by the system” would yield a huge and incomprehensible business model. There is thus a need for a horizon from which to understand the role of pervasive systems in assisting users in accomplishing what they want.

Breakdown as a partial mean for building a useful system

In the presentation of breakdowns earlier in this chapter it was hinted at that their occurrence in interactions with technology was a symptom of the technology not being tacit, as pervasive computing is envisioned to be.

Winograd [72] has treated the use of breakdowns in design of computer systems, but the view he takes is that “A breakdown is not a negative situation to be avoided, but a situation of non-obviousness, in which the recognition that something is missing leads to unconcealing (generating through our declarations) some aspect of the network of tools that we are engaged in using.”[72, p. 165].

The point here is that the state of reflection entered when a breakdown occurs leads to a better understanding of the task during which breakdown occurred and of the concepts playing a role in this task. The view underlying the currently proposed modeling technique is that this is true to different degrees for different breakdowns. In the example of hammering on page 64, where a breakdown occurred if the hammer’s head fell off we might ask how it is any good if this happens. It would be better if the hammer’s head stayed on and the nail was driven in with no problems at all. After all, who would like to live in a world where everything breaks down all the time.

The above illustrates that there are some breakdowns, more trivial in nature, which are better avoided because their occurrence is little else than in the way. This observation can be used to identify some tasks in which a pervasive system should assist, namely to avoid these trivial breakdowns, or at the least minimize their occurrence.
Modeling Primitives

The basic contents of the enhanced business model are still business actors, business use cases, business use case realizations and business objects. The additional part of this business model is a set of breakdowns. A breakdown is defined formally as follows:

**DEFINITION 4.6**

**A breakdown** is the occurrence of an event that is unforeseen and causes an interruption in the intended flow of some activity. The event arises from some entity that as a result of the emergence of the event draws attention to itself, and removes this attention from wherever it was before and thus primarily belonged. The attention drawn to the entity causes it to be perceived and thus, in the mind of the performer of the activity, emerge as an entity in itself.

A breakdown can be characterized by its name, the context it occurs in, a textual description, severity, and handling.

The name is descriptive of the breakdown, e.g. “Hammerhead falls off”. The context is the general activity in which the breakdown occurs.

The textual description follows the form of a detailed use case description, explaining the flow of events, with alternate flows, that the breakdown normally occurs in.

The severity is influenced by two metrics. Firstly an estimate of how much of an interruption an occurrence of the breakdown causes—its effect. This can be described either in terms of its immediate consequences, or if possible by the amount of time lost at its occurrence. The second metric of the severity is its frequency of occurrence.

The handling of a breakdown is either an avoidance or a recovery strategy. Both has the form of a brief sketch of ideas for how the breakdown can be either avoided or recovered from if it occurs. This is to be made more concrete later in the development.

The main structure of the enhanced business model is shown as a metamodel in figure 4.3. This metamodel only shows the parts of a business model as described in [27].

**Approach**

This section will explain how the enhanced business model is created and applied. In the enhanced business model, each of the parts in figure 4.3 need to be identified. The additional concept in this model is that of a breakdown. Breakdowns should by nature not be difficult to identify, since they by definition are interruptions that diverts the flow of attention in a given activity.

The following can be noted about how the breakdowns are employed in each activity in the development.
In **Business Modeling** the main breakdowns are identified along with the business use cases they occur in; this is done based on the vision of the system (the main idea held by those who suggested the creation of the system in the first place). Since the diversion of attention (that breakdowns are) is experienced only by the future users of the system it is vital that these are involved in identifying the initial set of breakdowns. Also, observation of their work may be necessary for an outsider to see the breakdowns.

In **Requirements capture** the strategies for handling breakdowns are refined. This is done by writing, in cooperation with the future users, the use cases, as is normally done. The additional issue that has to be treated in this activity is how the breakdown avoidance or recovery is experienced by the user.

In **the analysis workflow** the use case descriptions are treated as usual, but the actual workings of the breakdown avoidance or recovery strategy will manifest itself in the choice of how the use cases are realized. This means that the initial strategies specified in the business model are reconsidered in this activity.

In **design and implementation** the workflow is carried out as usual.

In **the test** workflow, it is of course verified that the functionality works as specified in the use cases. Additional to this, a user test (to the degree it is possible to perform) will be valuable to not only see if the already known breakdowns are handled in a satisfactory manner, but mainly because it enables identification of new breakdowns.

The use of breakdowns fit quite well with the iterative way of the Unified Process. This is because the most important breakdowns are those which occur in interactions with the pervasive technology, since these prevents the system from disappearing. Relative to iterations, the expected work with breakdowns goes like this:
4.4 Conclusion

The purpose of the activity described in this chapter was to obtain a modeling technique for pervasive systems. With respect to that goal, the following can be noted:

- The criterion 2.1 page 43 stated that it should be possible to designate which classes in a model has a physical dimension, that is, whose instances are tangible. This has been done by introducing a new stereotype, “tango-class” which is a class whose instances are tangible objects. What is accomplished by doing this lies mainly in the introduction of the concept of a tango-class, since its notation is standard UML.

- The criteria 2.2 page 43 stated that there was a need for a frame of reference for reasoning about the physical aspects of the system. This issue has been addressed by introducing a new kind of diagram (a map) and a set of concepts and notation for creating these maps.
Criterion 2.3 page 44 requires a suitable model for identification of use cases, and a clearly defined approach for creating this model. To accomplish this, it was suggested to employ *breakdowns* to partly specify with what the system should assist its users. These were incorporate into an “enhanced” business model. The creation of this model (mainly the identification of breakdowns) should be done initially in cooperation with the future users, and in subsequent iterations it should be refined by identifying additional breakdowns through user tests.

It can thus be concluded that a step has been taken towards being able to model a pervasive system.
Chapter 5

The Interaction Widget Pattern

This chapter describes the pattern *Interaction Widget*. It was hatched alongside working on the main activities planned for this project.

5.1 Problem

When a user looks at an application, he or she sees the user interface; in every sense that matters, the user interface is the application from the user’s perspective. When a developer looks at an application, the user is mainly seen as a source and destination of information. The user interface should thus be made sense of in different ways from different perspectives.

Consider the user interface of DOS. It provides no visual guidance for the user. Essentially, everything the user does has to be entered as an array of strings, a form of input that is rather close to the developer’s way of seeing a user action (shells are still used a lot —by developers). Many programs made during the period where DOS was prevalent provided a better user interface than DOS, some did little beyond providing a better UI to DOS (e.g. Norton Commander, Shell). But in order to make these interfaces, the developers needed to build each interface from scratch. This meant drawing the interface as arrays of ascii characters on the screen, and making the code more complex so that it could deal with registration of user actions and updating the interface.

Building an interface from the ground up is a complex task that requires a lot of work, and even when it is complete and working, its appearance and programmatic interface is still not standard, and cannot be reused by others. A developer, even one familiar with building user interfaces, would have a difficult time understanding the interface code written by another developer. So for developers of DOS applications, the main problems was lack of a way to reuse code and the difficulty in understanding the code written by others.

---

1The pattern described here has been accepted for publication as: *Interaction Widgets*. Ingstrup, Mads. Proceedings of the 8th European Conference on Pattern Languages of Programs, June 25th – 29th 2003, Irsee, Germany.
In the days of DOS, standardization of the user interface could not be relied upon, which was at the expense of the users, who had to work differently from application to application. That each interface had to be built from scratch also meant that their quality varied in terms of ease of use and functionality. So it may be fair to say that both the users and the developers suffered as a result of a lack of bridging between their two perspectives.

From the perspective of the user, the following forces are at play:

- The users desire a standardized interface through which interactions take place. This enables them to familiarize themselves with the interface, not just for a single application but across different applications.

- Interactions should be understandable for the users and the interface should guide them through the use of the application. This is especially important for new users. Part of making the user interface understandable is to enable the user to interact with it using user interface elements that he or she can understand or is familiar with. These include such artifacts as: mice, double-clicking, arrow-keys and respective behaviours.

From the perspective of the developer, the following forces are at play:

- Designing interactions and interfaces is difficult, so developers should be able to reuse proven ways of channeling input and output between the user and the system. The developers should be insulated from having to deal with the complexities of low-level interactions like capturing events and updating the screen.

- Interactions should be understandable for the developer in that it should be clear what information in what format is exchanged. The developer should be able to access and handle this interaction in a way that he or she is familiar with; a standardized programmatic interface. These means include concepts such as method invocations, use of operators and datatypes. For instance, in the Java Api this is done by defining interfaces such as java.awt.event.ActionEvent and requiring this to be implemented by classes whose instances should be notified of events in widgets like java.swing.JButton. Another principle used in Java Api is the convention of prefixing method names for reading an objects value with get, e.g. getPercentComplete() in java.swing.JProgressBar.

5.2 Solution

We need to bridge the gap between these two perspectives in the creation of a user interface. To solve the problem, bridge the gap in perspectives by using widgets to mediate this shared information. A widget is an element of the user interface that the user can interact with—such as a button, scroll bar, canvas, window. A
5.2 Solution

A widget encapsulates the behaviour of that user interface element. It hides the user’s perspective from the developer, and the developer’s perspective from the user.

What a widget does is to capture the essence of a part of an interaction and present different abstractions of this part to the user and the developer. To the developer the abstraction hides the concrete mechanics and complexity of the implementation (e.g. how and when the screen is updated) and it obeys a standardized interface in order for it to be reusable, c.f. the last force in the developers perspective above. For the user each widget must capture the nature of the information it should mediate by preventing wrong information from being entered or visualized. For instance, the way radiobuttons prevent more than option from being selected.

A widget consists of:

- a *mechanism* that enables capturing or presentation of information. (e.g. a button can be clicked on).
- a *name* that is descriptive of the widget’s behavior and characteristics. (e.g. “button” is descriptive of the behavior and characteristics of physical buttons).
- a *representation* that matches and helps shape the user’s mental model of that widget and its behavior. (e.g. a button widget in a GUI uses a stylized pictorial representation of a button, an icon for it).
- a *standardized interface* through which the functionality of the widget can be accessed and used by a developer. (e.g. a class java.swing.JButton with methods and attributes accessible to the developer).

From the perspective of the user and the developer, an important thing is invariant. This is the understanding of the information and actions that are exchanged in the interactions. For instance, a button named Save will be associated with the meaning that something is stored permanently, and this meaning is shared by the user and the developer. The two perspectives and their overlap is illustrated in the figure below.
EXAMPLE 5.1
The buttons that are used in GUIs are examples of widgets. They mediate information both to and from the system. Firstly, a button presents an action to the user through its label, e.g. “Open”. The meaning of the action is also conveyed through the position of the widget in relation to other widgets, for instance in the image below, it is clear that it is the file by the name appearing in the textfield that is opened.

Secondly, it allows the user to perform an action since it through being clickable allows the user to express his or her intent to perform this action. From the user’s perspective, the button resembles a physical button. This is significant because it helps convey the button’s affordance of being clickable by establishing a connotation to a physical button whose affordances the user is familiar with. Note the use of standardization here, where the underlined “O” in Open means that the action can also be performed with the shortcut «Alt+O».

From the developer’s perspective, the button is an object that can fire notifications to an observer. This functionality is hidden behind an interface and accessed by methods such as addActionListener(...), in the case of Java buttons. This means that the developer need not worry about the button’s implementation and the conformance with a clearly defined interface enables reuse of the button. What is shared by the user and the developer is the meaning of the information being mediated, in this case the meaning of the action that the button lets the user perform.

5.3 Consequences
Use of the pattern has the following benefits:

- By looking at and interacting with a widget, a user can derive its affordances. This makes it easier to understand in what way the widget should be used. For instance, the idea behind a scroll bar may be comprehended by trying to use the mouse on the parts of it that resemble buttons. The widget can also preclude the entry of information in the wrong format, structuring the user’s access (e.g. radio buttons do not allow more than one selected option).

- Because widgets are standardized and encapsulate their behaviour, a developer can reuse the widgets and save time in developing ways of interaction with the system.

- The specifics of the interaction are hidden from the developer, thus insulating the application: as noted by [56], “Whether the user points and clicks with a mouse ... or uses keyboard shortcuts doesn’t require changes to the
application.” So the widget provides the developer with abstracted information (effectively “this action was performed” rather than “the user used the mouse to click at coordinate (x,y)”.

When applying the pattern, be aware of the following:

- If widgets are not designed carefully, they might be difficult to use, especially for new users. The lesser the degree to which a widget conveys its affordances through its appearance, the more important standardization becomes (by the same argument that non-blue non-underlined weblinks reduces usability). For instance, many widgets found in GUI toolkits can be disabled, often shown by the widget being “greyed out”. The use of “greying out” is an example of a standard screen metaphor that is used for indicating disabled widgets to the user.

- Also, from the developers perspective, if a widget is not designed in a predictable way, it may be difficult to use. Specifically, once a developer has found the right widget for some purpose, it may not be easily apparent how to extract information (text, numbers, selections) from it—especially if the programmatic interface for it does not follow any conventions or otherwise is non-intuitive. For instance, a developer will almost certainly know that a String object can be obtained from a text field widget, but may be in doubt as to how this is done. So care must be taken to obey the standard style and idioms used in formatting an interface (for instance, in Java classes, methods for getting a value of a field should be called get\(<\)FieldName\(>()\) )

- Depending on how an interface is designed, widgets may slow down the user’s interaction with the system. For instance, a highly GUI-based, mouse-driven travel reservation system may slow down a booking agent, who may be used to text-based, keyboard-intensive entry (a historical artifact of early systems). Thus, the type of widgets that are used for an interface should take into account the types of users who will be the target audience of the application.

- Finally, widgets are not silver bullets. Widgets offer advantages, but a developer will still need to use their functionality and combine them in a way that is usable and useful to the user. For instance, a change in the state of a program should be visible to the user. The selection of what actions are possible at any given time should also be considered carefully. The widgets should be spatially grouped in a sensible way (e.g. all buttons for selecting alignment of paragraphs of text in a word processor should appear near each other). In other words, the usability design of a user interface is not automatically resolved by this pattern, but interaction widgets may provide a basis for reasoning about some of these issues, like the one with spatial grouping of widgets.
5.4 Known Uses

Widgets are used widely in GUI toolkits, taking such forms as buttons, radio buttons, check-boxes, windows etc.

The emerging discipline of Pervasive and Ubiquitous computing introduces a new, physical dimension compared to more traditional systems. This does not preclude use of the pattern. To ease the construction of user interfaces for ubiquitous systems, the Context Toolkit [56] defines a number of interaction widgets that mediate between the environment (including users) and the application in the same way as graphical widgets mediate between the user and the application. An example of an interaction widget from the toolkit is the "Activity Sensor", designed to register when there is activity in a given room. It consists of physical sensors, e.g. an activity detector, and a software interface/class. Here the pattern extends into the physical dimension, but the forces remain the same.

Another example of a currently emerging class of widgets is in the area of biometric identification [28]. One widget may scan the fingerprint of a user, another may scan the iris of the user’s eye, but both deliver the same abstract information to an application, namely the verified identity of the person in front of the widget.

5.5 Related Patterns

The Model-View-Controller architectural pattern [8] shows how an application can be divided into Model, View and Controller components. This pattern is different in the following respects:

- The perspectives differ. MVC is an architectural pattern focusing on how to build an application from the perspective of the developer; it is a pattern for a whole application. This pattern employs both the user and developer perspective and seeks to resolve forces perceived from both of these perspectives; it only focuses on the interface part of the application.

- Both patterns argue building the interface of blocks (views and widgets respectively), but there is a difference in scale. Widgets are more fine-grained such that a view may be made of several widgets.

- The patterns complement each other with Interaction Widget describing the part of building views that is left out in MVC. Interaction Widget may be used equally well together with the PAC architectural pattern [8]. This is because both PAC and MVC are concerned with the internal structure of the system, but advocates that the interface is kept separate. The use of interaction widget, pertains only to the interface. And, the issues it address are not addressed in either MVC or PAC, and will therefore not interfere with or contradict anything prescribed by these two. One might say that doing as prescribed by Interaction Widget does not cross any of the boundaries set up in the division of a system prescribed by these two.
Appendix A

Reflections on the observations of activity one

A.1 Micro iterations

Within UP, the concept of iterations is central. But it was noted that within the iteration that was performed, minor revisions was made in an iterative fashion as well. For instance if it at a given point was found necessary to make a minor change in BM, then this often required a change in the RM as well. Thus, conducting these changes might start with a small adjustment in BM, then RM, and perhaps finally AM. The work of tracing and adjusting such changes through two or more models will here be called a microiteration.

Example A.1
An example of this is that when BM was made, initially there was no generalization between the Reserve Resource and Reserve Meeting Room. Of course it quickly became clear that these two was almost identical in their flow of events descriptions. So once the initial draft of RM was made, the business use case model was changed to reflect the similarity between these two use cases. Then the use cases in RM (bearing the same names) that could be traced to these two business use cases were also changed in order to reflect their similarity, in fact they were merged into one, slightly more general use case (Reserve Resource).

A.1.1 Reflections on method of work

The observations made about the practical work, e.g. micro iterations, does not match with the prescribed method of work in the Unified Process. To explain the inconsistency, consider the three kinds of methods described by Raccoon\(^1\) [51] who explains them using a figure such as A.1.

\(^1\)This is not the real name of the author, who for unknown reasons has choosen to remain anonymous. The article however speaks for itself.
Figure A.1: This figure is taken from p.95 in [51]

In the Waterfall model [?], the process of developing software is divided into the phases requirements capture, analysis, design, implementation etc. Each phase is performed once, and the resulting model remains fixed for the duration of the project. The Spiral model is iterative, but each iteration is in a sense an application of the waterfall approach. This is how the Unified Process is presented in [27]:

But, an iteration is not an entirely independent entity. It is a stage within a project. It draws heavily from being part of a project. We say it is a miniproject because it is not, by itself, what the stakeholders have asked us to do. Also, each of these miniprojects is like the old waterfall model because it proceeds through the waterfall activities. We might label each iteration a “miniwaterfall.”

The notion of micro iterations captures to some extend the manner in which the practical application of UP was performed, but the reverse change effects, described in section ??, are only recognized in the Chaos model.

A.2 Reflection on results

A.2.1 Regarding shortcomings detected during analysis

What can be said of no. 1–5 is that they can all be solved by adding new use cases to RM. And, it would seem that these use cases are all secondary. What the shortcomings imply are in all cases a change to RM caused by observations made in AM. The AM is the first view of the internals of the system, and thus of how the system deals with the entities of the problem domain, and the entities the has been introduced in the analysis workflow itself. Further, in every case observed, the actual cause that the shortcomings were detected seems to be that this internal view of the system, that is AM, were in a sense not complete. In other words, when considering the treatment by Tais of some entity, say Resource, then it is only natural to think of how these are created. This is because it is a part of...
the analysis workflow to analyze the classes, which includes to specify where and when each class is created, and how long it exists.

Changes of this nature is illustrated in figure A.2. When an arrow goes from a use case to a class it means that the class is used to realize the use case. When a class points to a use case (be it there or not) it means that the class should be used in this use case, e.g. that it should be created, scheduled, or have something else happen to it in this use case. The experience of applying UP has been that the dependencies represented by the arrows from top to bottom are used as a means to identify the analysis classes. But, as exemplified by the observations, a class can also require the presence of certain use cases, those that are needed to manage the instances of this class. These are by nature secondary.

### A.2.2 Primary and Secondary Use cases

Considering the observations presented in section 1.5.1 it seems that a change in a business use case in BM causes changes mostly to the primary use cases, whereas the secondary use cases are more dependent on what entities are in the problem domain; those described in the business object model.

Regarding the creation of these two types of use cases, a picture is thus formed where the primary use cases seem to be specified mainly from the business use cases. The secondary use cases in RM then arise from the business use cases in an indirect manner, as a form of minimum functionality required for the areas that the business use cases are concerned with; the problem domain. The following hypotheses sums this up:

**HYPOTHESIS 13**

*The primary use cases are to a large degree specified from the business use cases.*

**HYPOTHESIS 14**

*The secondary use cases specify the minimum functionality needed in a system for a given problem domain.*
Consider then how the functionality specified in these two categories of use cases were distributed into analysis packages. The primary use cases were implemented in the packages at the application specific layer. The functionality specified in the secondary use cases were implemented in the packages at the application general layer.

The functionality in the application general layer, e.g. handling of calendars, accounts, resources may well be captured in frameworks because of their general nature. This seems to be the case even for business rules, although this subject is somewhat more complex. In contrast, that which is peculiar to the TAIS is in the package at the application specific layer, where there by nature are fewer opportunities for building reusable application frameworks. So, although more would be needed to verify it, the following hypothesis may be stated:

**HYPOTHESIS 15**

*The secondary use cases are more relevant to the development of frameworks than the primary ones.*

As a final remark, note that when considering framework development it is a common problem to decide how general/specific the framework should be. This goes well with the observation that rather than the primary and secondary use cases being sharply separated groups, they are the two ends on a continuous scale. Thus, for a given use case it may be difficult to say whether it is primary or secondary.

### A.2.3 Comparison of properties of BM, RM, AM (DM)

One aspect of the overall experience of working with UP, may be stated in the following hypothesis:

**HYPOTHESIS 16**

*The work of creating BM and RM are done in a bottom-up fashion, whereas the work of creating AM and DM is done top-down.*

The BM presented here was made for an existing organization, the Maersk Institute. It was done by interviewing mainly the two secretaries using questions like “What should I do to get to use the projector?” and “What is done from the point on deciding to hold a board meeting untill it has taken place?”. The answers, which consisted of what came to be steps in the business use case were then grouped into use cases of adequate sizes. The roles of the people who could take part in these steps were then ordered into the roles represented by the business actors. Alongside doing this the interdependencies of the actors and use cases were developed, with generalizations being done after having the use cases in their first drafts. The business object model were easily put together after having described the use cases. So the work of creating BM were bottom-up starting with the individual steps in the use cases, the individual persons that may take part in these steps, and ending with a structured model of the use cases and actors.
Appendix B

Activities and Calendar

Activity I

*Application of UP to a traditional administrative system*

This activity will use The Unified Process, [27] (henceforth denoted UP) in the development of a traditional administrative system. The domain will be an academic institution, namely the Maersk Institute, but an aspect of it that it is realistic to have supported by a pervasive system.

**RESULTS**

The concrete results are business model, requirements, analysis, and partly design documents of the system. A less concrete but equally important result is that by performing this activity a familiarity with the problem and application domains of systems for the institute are obtained.

**METHOD**

Straightforward application of UP.

Activity II

*Application of UP to a pervasive system in the same domain*

The system developed in the first activity will be extended in this one. The extensions are to make the system pervasive. How to specify requirements for a pervasive system is a subject in itself, but the imagined way of doing so is by augmenting existing use cases and adding new ones.

By letting this activity be concerned with the same system as that of the first, it becomes possible to reason about which problems occurring in this second activity are due to the system being pervasive, as it must be those not occurring at the same point in the first activity.
RESULT
A description of the problems encountered when trying model a pervasive system in the way prescribed by UP, but only those arising from the system being pervasive. Based on these observations, a set of criteria for a modeling technique for pervasive systems.

METHOD
Straightforward application of UP (or an attempt of such). Variations from the activities prescribed by UP are not to occur, so if for some reason an activity cannot be performed, and this was not the case for the corresponding situation in activity one, then a description of the problem is noted as a result of this activity.

In both this and the previous activity special focus should be dedicated to those aspects of UP concerned with modeling. This is because the overall goal is to infer a new way of modeling. A consequence is that a noted problem will, provided it is concerned with modeling, not only be reported, but also analyzed to determine its causes. Another consequence of this special focus is that the part of the analysis and design documents accounted for in this report will primarily be those concerned with modeling.

Activity III
System vs. Organization relationship

This activity will investigate how the organization of the users can be used as a source of input to the process of creating a software system. As has already been argued, this is seen as being of increased importance in pervasive systems. But it is also of importance in more traditional systems, as indicated by the studies [44, 45]. These show that, in the case of intranets, the technological skills alone are not enough to create the right system. What is lacking is that the softer issues in the application and problem domains are taken into account, according to [44]. In relation to the view of organizations depicted in figure i.3 page 11 all the forces that fashion the structure of an organization can also influence the requirements to or success of a software system.

METHOD
Mintzberg [43] has proposed five stereotypical organizations. They can be considered patterns of organizational structure, describing what structure emerge when different factors dominate the organizational setting. These stereotypes will be analyzed to uncover which of their characteristics are likely to influence how a software system should be shaped. A similar idea has been pursued by [35, 30], although the goals of these were to understand information systems rather than to develop software. Here, the method is to analyze how the forces resulting in a given organizational structure will work to emphasize the importance of particular qual-
ities in the software. To set the stage for this analysis, software and organizations will be compared. This serves the purpose of investigating how alike software and organizations are, and thus whether the same forces working on both can in any way be expected to yield similar effects.

RESULTS
The results of this activity will be presented by how it relates to the stereotypical organizations. It is described what it is important to be aware of in the development of software for each of them. Since different mechanisms and functions dominate the individual stereotypes, the guidelines may well vary in both quantity and flavour, from emphasizing important nonfunctional characteristics to concrete architectural patterns to more or less general design philosophies.

Activity IV

A Modeling Technique for Pervasive Systems

The results of activity two should disclose a number of difficulties with applying UP to development of pervasive systems. Activity three shows, given an organization, what is the nature of its influence on software development. This activity will take the results of the previous activities into account when creating a modeling technique for pervasive systems.

RESULT
The main result of this activity will be a modeling technique for pervasive systems. That is, a proposition of some terms (modeling primitives) that are suitable for expressing and reasoning about a pervasive system. Also part of the technique is the way in which they are used.

METHOD
The work in this activity includes two major parts. Firstly, to derive a number of criteria by which a modeling technique may be judged. Secondly, to propose a modeling technique that will adhere to these criteria.

Activity one and two will provide concrete examples of some modeling problems that occur in trying to use UP and UML to build a pervasive system. That is, in the “language” of the modeling primitives, it should be possible to articulate the concepts that are at play in the observations and criteria made in activity II.

To create a modeling technique, the first step will be to study the philosophy of modeling. This will discuss such questions as what modeling is, what we do when we model etc. Based on this and on the criteria, a modeling technique will be proposed. The method in this activity explained more elaborately in 4.1 page 61.
Use modeling technique

The purpose of this activity is to test the obtained modeling technique on a real case. Again, the system chosen is that of activity two, but only the part of the development that could not be performed during this. That is, the original problems of modeling will be used as a test.

RESULT

An evaluation of the aptness of the modeling technique for modeling the pervasive system specified as part of activity two. Having used the obtained modeling technique to take care of the first problems, others may become visible.

METHOD

Use of the modeling technique as prescribed by the result of activity four, on the problems described as the result of activity two.

B.1 Project Calendar

This is a schematic mapping the activities and their duration onto the timespan set aside for the project.

<table>
<thead>
<tr>
<th>Week no.</th>
<th>40</th>
<th>41</th>
<th>42</th>
<th>43</th>
<th>44</th>
<th>45</th>
<th>46</th>
<th>47</th>
<th>48</th>
<th>49</th>
<th>50</th>
</tr>
</thead>
<tbody>
<tr>
<td>Project Planning</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Activity I</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>Activity II</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>Activity III</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
</tr>
</tbody>
</table>

| Week no. | 5  | 6  | 7  | 8  | 9  | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 |
|----------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| Activity II |    |    |    |    |    |    |    |    |    |    |    | x  |    |    |    |    |
| Activity III | x  | x  | x  | x  |    |    |    |    |    |    |    |    |    |    |    |    |
| Activity IV   |    |    |    |    |    |    |    |    |    |    |    | x  | x  | x  |    |    |
| Activity V    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    | x  |
| Finishing the report |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    | x  |

Activity no. IV includes a study of the philosophy of modeling which is performed as literature is found and becomes available.
Appendix C

Properties of pervasive computing

This table is taken from [40]

<table>
<thead>
<tr>
<th>Property</th>
<th>Pervasive computing view</th>
<th>Traditional computing view</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intention</td>
<td>Objects and habitats should have an intention, purpose or goal. This is necessary to support autonomous and intelligent behavior, especially when supporting the contextual requirements of humans.</td>
<td>Objects and environments are designed to perform a function. This function does not assume intention or goal, especially in the context of intelligent behavior or human requirements.</td>
</tr>
<tr>
<td>History</td>
<td>Objects and habitats should remember what has happened to them over time, building up a memory of their experience and their interaction with the external world. This is important for supporting adaptive behavior in objects and habitats.</td>
<td>Objects and environments don't inherently build up a history of their experience; they are aware of their current execution and state but not what led up to that point. Objects and environments only conceive of the present.</td>
</tr>
<tr>
<td>Dynamic</td>
<td>Objects and habitats should be able to participate in interaction whose structure is not fixed and pre-defined. Interaction and structure be dynamic, allowing flexibility.</td>
<td>Objects and environments exhibit dynamic behavior, but in most systems, the structure of interaction tends to be fixed and pre-defined. Dynamic structure is more of an exception rather than being de facto.</td>
</tr>
</tbody>
</table>

continues on next page
<table>
<thead>
<tr>
<th>Property</th>
<th>Pervasive computing view</th>
<th>Traditional computing view</th>
</tr>
</thead>
<tbody>
<tr>
<td>Personalized</td>
<td>Objects and habitats should personalize their interaction with other objects and habitats and humans. This requires some knowledge of the history of an interaction.</td>
<td>Objects and environments do not personalize their interaction; interaction is assumed to be homogeneous.</td>
</tr>
<tr>
<td>Adaptation</td>
<td>Objects and habitats should learn from their past experience and adapt their existing and future behavior. The history property supports tracking of experience while the dynamic property supports the act of changing behavior.</td>
<td>Objects and environments by default do not keep track of history, nor do they have the ability of adapting their behavior autonomously over their lifetime.</td>
</tr>
<tr>
<td>Anticipation</td>
<td>Objects and habitats should anticipate and foresee future interaction that might happen. The history property supports objects and habitats in their review of what has happened and what might happen.</td>
<td>Objects and environments do not anticipate future interaction; they tend to only conceive of the present.</td>
</tr>
<tr>
<td>Time and space</td>
<td>Objects and habitats should have a notion of physical time and space as it relates to (a) other objects and habitats (b) the real world. This is to support interaction with entities that may be working with different time scales in different habitats.</td>
<td>Objects and environments do not have notions of physical time and space unless they are designed for specific application domains. Objects and environments tend to be independent of physical time and space considerations.</td>
</tr>
</tbody>
</table>


Index

TAIS, 23
acceptability, 55
adhocracy, 58
analysis model, 25
application domain, 2
breakdown, 62
business model, 24
business rules, 27
constraints, 63
convergence, 6
design philosophy, 4
development process, 3
divisionalized form, 56
frameworks, 78
function component, 3
Heidegger, 62
hypermedia, 5
interface component, 3
location modeling, 66
machine bureaucracy, 52
micro iterations, 75
model component, 3
pervasive computing, 5
problem domain, 2
professional bureaucracy, 54
radical tailorability, 5
requirements model, 24
simple structure, 52
stereotype, 63
tagged values, 63
tangible object, 64
thrownness, 62
ubiquitous computing, 5
UML, 63
use case
  primary, 30
  secondary, 30
use case model, 24

Weiser, Mark, 5
APPLYING the Unified Process in the design of the Maersk Institute Administrative System

Mads Ingstrup

This report contains together with a model in Rational Rose the documentation of the results of applying the Unified process to the administrative system for the Maersk institute. It thus documents activity one in the array of activities described in the msc report.
Table of Contents

1.0 Requirements 3
   1.1 Introduction 3
   1.2 Business Model 3
      1.2.1 Business Actors 3
      1.2.2 Business Use Cases 3
      1.2.3 Business Object Model 6
   1.3 Use Case Model 6
      1.3.1 Actors 6
      1.3.2 Use Cases -- Brief Description 7
      1.3.3 The use case model as a whole 8
      1.3.4 Architecture Description/Prioritizing the Use Cases 9
      1.3.5 Detailed Description of each Use Case 10

2.0 Analysis 14
   2.1 Architectural Analysis 14
      2.1.1 Analysis Packages 14
      2.1.2 Obvious entity classes 15
      2.1.3 Common Special Requirements 16
   2.2 Analysis of Use Cases 16
      2.2.1 Schedule Meeting 17
      2.2.2 Arrange Board Meeting 18
      2.2.3 Submit Agenda Points 20
      2.2.4 Reserve Resource 20
      2.2.5 Create/Edit Business Rule 22
   2.3 Remaining Use Cases 24
   2.4 Analysis of Classes 24
   2.5 Analysis of Packages 27

3.0 Missing Features discovered in First Iteration 27

4.0 Design 29
   4.1 Architectural Design 29
      4.1.1 Subsystems and their Interfaces 30
   4.2 Architecturally Significant Design Classes 31

5.0 References 31
1.0 Requirements

This is documents the workflow of gathering requirements cf. the workflow described in section 7.4 in [1].

1.1 Introduction

The system that is to be developed is an administrative system for use at the Maersk McKinney Møller Institute for Production Technology. A number of administrative problems currently exists, and it is believed that a system can be made that will help resolve these. The primary tasks of the system will be:

- To manage the reservations of meeting rooms, projectors, and other resources.
- Maintain a calendar for the staff, thereby easing the task of finding a time for a meeting that does not conflict with the plans of any of its participants.
- Assist in making available agendas for and minutes of meetings.

1.2 Business Model

A business model consists of a number of artefacts. These are actors, use cases, use case realizations, and a set of business entities and work units participating in the use case realizations. Note that in the following diagrams, a line (/) in the symbols of use cases, actors and classes means that they are the business variant of the stereotype. (For instance, a business use case is a use case describing how a business actor uses the business the same way a normal (not business) use case describes how a normal actor uses the system.)

1.2.1 Business Actors

The following business actors appear in the business use cases.

Official
A person which holds some formal title in relation to the institute. This includes every employee, but also student representatives in various committees.

Secretary
Is a specialization of Official, and a person responsible for writing and publishing agendas for and minutes of formal meetings such as board meetings. The actual person playing the role may vary, for instance it is played by a secretary for meetings in the internal board while the director of the institute is Secretary for the meetings in the external board.

1.2.2 Business Use Cases

The following business use cases are considered relevant for the system.

- Arrange Board Meeting
- Reserve Meeting Room
- Schedule Meeting
- Reserve Projector
The Business Use Case Model as a Whole

The Secretary may use the *Arrange Board Meeting* use case to call for points to the agenda of the meeting, when a board meeting has already been scheduled using *Schedule Meeting*. The *Arrange Board Meeting* requires the assistance of several Officials, who are the participants in the meeting and thus has the right to put points on the agenda.

Turning to the use cases that can be initiated by an official, the *Schedule Meeting* is used to find a date and time for a meeting when it’s participants are known. The *Reserve Projector* and *Reserve Meeting Room* are used by an Official to reserve a resource, for a given period of time at a given date and time.

Below follows a description of each of these use cases, in terms of their flow of events and pre- and postconditions.

**FIGURE 1.** Business Use Case Diagram

![Business Use Case Diagram](image)

**Arrange Board Meeting**

*Precondition*

The date and time of the meeting has already been decided.

*Flow of events*

1. The secretary sends an email to the participants. The mail reminds of the date and time of the meeting, and contains a request for submission of points to the agenda.
2. Each official replies, through email to the secretary, with relevant points.
3. Upon receiving the answers, the secretary writes the agenda and mail it to each participant.
4. When the meeting is held, the secretary takes notes, and uses these when writing the minutes of the meeting.
5. The minutes of the meeting is mailed by the secretary to the participants in the meeting.

Postcondition
The meeting has been conducted, and the minutes are available to each of its participants.

Schedule Meeting

Precondition
A reason for having a meeting have arisen, or it is time to arrange a meeting of a kind that is scheduled regularly (like a board meeting once a month).

Flow of events

1. An official initiates the meeting by inviting the intended participants, and by email requesting them to get to reply what times they are busy.
2. By comparing the replies, a time is found where the highest number of participants will be able to attend.

Postcondition
A time has been allocated for the meeting.

Reserve Projector

Precondition
A projector is needed.

Flow of events

1. The official checks with the calendar for either of the projectors until one has been found that is available on the given time.
2. The official enters a reservation for the projector in the calendar associated with it.

Postcondition
A projector has been booked, or none of them was available at the needed time.

Reserve Meeting Room

Precondition
A meeting room is needed.

Flow of events:

1. An official looks in the calendar for the meeting room, and decides on a time where the meeting room is available.
2. The official writes an entry into the calendar, thus reserving the meeting room for a given period. The entry includes the officials contact information (e.g. login).

Optional Flow
If the meeting room was already reserved, but the purpose for the current reservation attempt has higher priority, then the previous booking is cancelled and the official who entered it is notified.

Postcondition
The meeting has been booked, or it was busy with a higher prioritized task.
1.2.3 **Business Object Model**

Below a diagram of the business entities and their relations are shown.

**FIGURE 2. Business Entities**

![Diagram of business entities and relations](image)

Find Actors and Use Cases

### 1.3 Use Case Model

#### 1.3.1 Actors

The following actors have been identified.

**User**

The User actor in the use case model can be traced back to the Official actor in the Business Model. The role of User is thus played by everyone who plays an active role at the institute, ranging from tasks such as participating in meetings to using resources at the institute.

**Secretary**

The Secretary Actor in the use case model can be traced back to the Secretary actor in the business model. The Secretary role is thus played by everyone who uses the system for those tasks performed by the business actor equivalent. Conforming with the Business-Secretary being a special kind of Official, this Secretary is a specialization of the User actor.

**System Administrator**

The system administrator is responsible for maintaining the system, and has the right to change priorities, create user accounts etc. Being a specialized form of user this actor can be traced from the Official business actor.

**Manager**

A manager is someone who has an administrative position high enough to enable him or her to decide and implement business rules. Manager is a specialization of User.
1.3.2 Use Cases -- Brief Description

Currently, these use cases have been identified:

**Arrange Board Meeting**
- is used by a Secretary whenever a board meeting has been scheduled, and points for
  the agenda needs to be requested from the participating officials.

**Schedule Meeting**
- is used by a User to find a date and time where the highest number of participants
  will be able to attend.

**Submit Agenda Points**
- is used by a User to submit points for an agenda for a particular meeting.

**Reserve Resource**
- is used by a User to reserve a resource for a given period of time at a given time and
  place.

**Login/Logout**
- is used by the User to log on, respectively off, the system.

**Create/Edit Entity** (abstract)
- is used by a User to create or edit an entity. The use case is abstract.

**Create/Edit Account**
- is used by the System Administrator to create or edit a user's account. The special-
  ization from Create/Edit Entity lies in the user being a System Administrator, and
  the Entity being an account.

**Create/Edit Business Rule**
- The Manager uses this use case to add new business rules or to edit existing ones.
  These may include: (1) criteria for suggesting meeting dates, according to priority of
  participants. (2) rules for when a booking of a resource may be overridden by
  another user, depending on this users priority or the priority of the task for which the
  user intends to reserve a resource (i.e. an exam has high priority, but a student cannot
  book a room for such an event, as students do not perform examinations).
  The specialization from Create/Edit Entity lies in the User being a Manager, and the
  Entity being a business rule.

**Delete Entity** (abstract)
- This use case is used by a User to delete an entity from the system. The use case is
  abstract.

**Delete Account**
- This use case is used by the System Administrator to delete an account from the sys-
  tem. It is a specialization of Delete Entity, and the refinement lies in the User being a
  System Administrator, and the entity being an account.

**Delete Business Rule**
This use case is used by a Manager to delete a business rule from the system. It is a specialization of Delete Entity, and the refinement lies in the User being a Manager, and the entity being a business rule.

1.3.3 The use case model as a whole
In Figure 3 on page 8 a use case diagram shows all the use cases and actors and the relations among these. (Reserve Meeting Room and Reserve Projector are only shown as illustration of how Reserve Resource is derived from the business use cases; they seem similar enough that any distinction is superfluous).

Starting with Arrange Board Meeting the secretary may use this use case to call for points for the agenda of the meeting. As for the corresponding business use case, this can only be performed once the board meeting has been scheduled using Schedule Meeting.

Moving on to the ordinary User, any use of the system by this actor (and others who are specializations of it) starts with logging on to the system by means of Login, and ends by logging off the system, with Logout. The User may now use Schedule Meeting to find a time and date for a meeting where a suitable number of participants will be able to attend. (By Suitable is meant the clarified in the next paragraph). Once a meeting has been scheduled, and the user in question is invited (this happens as part of Schedule Meeting, where each participant is notified of the time and place for the scheduled meeting), the User may receive a request for submission of points to the agenda. Performing this submission can be done in Submit Agenda Points.
The System Administrator will use the system to manage accounts, creating/editing and deleting them with Create/Edit Account and Delete Account respectively.

Finally, the Manager has the right to change and add business rules using the Create/Edit Business Rule, and Delete Business Rule use cases.

The similarity in the flow of events between the initial Create/Edit Business Rule and Create/Edit Account led to the more general (abstract) use case Create/Edit Entity, from which they both inherit. The same is the case for the use cases concerning deletion of entities.

1.3.4 Architecture Description/Prioritizing the Use Cases
(This section has not been updated. It should be altered according to chapter 12.6 in [1])

The prioritization of the use cases has been done by the following criteria. Firstly, if two use cases concern the same parts of the system, only one of them has been given a high priority. Secondly, some weight is given as to how use cases depend on each other, in terms of testability. Thirdly, it is of influence how widely the system is affected by the functionality specified in the use case, that is, how architecturally significant it is.

Based on this, the priority of the use cases is the following.

1. Create Account
2. Edit Business Rules
3. Login and Logout
4. Schedule Meeting
5. Arrange Board Meeting
6. Reserve Resource
7. Submit Agenda Points
8. Delete/Modify Account

Firstly, Create Account is important because the accounts of users are accessed in all of the remaining use cases.

Edit Business Rules is second most important because it will is concerned with the rules that may be used in most of the other use cases (each of no. 4--7), so reaching a stable implementation of these rules is vital.

Login and Logout are important for reasons of testability, since they are necessary to test system functionality that involves multiple users. Also, since the amount of work required to implement them is limited compared to most of the other use cases, it does no harm to have them implemented at an early stage.

Schedule Meeting is prioritized next, because it specifies how messages are to be send between individual users, and because it requires the calendar matching functionality, which is considered important as well (used in Reserve Resource).

The remaining three are almost of equal importance, but Arrange Board Meeting is given the highest priority because it must prelude the test of Submit Agenda Points.
Requirements

Reserve Resource specifies a larger extension than does the Submit Agenda Points which thus has lower priority of the two.

Finally, Delete/Modify Account is given the least priority because, first of all no other use case depends on it, and because the functionality it specifies is thus relatively separate from the rest of the system.

1.3.5 Detailed Description of each Use Case

Each of the descriptions given below follows the form already applied in the descriptions of the business use cases.

Create/Edit Entity (abstract)

Precondition

The entity to be edited exists already, or if an entity is created, an equivalent one (same user login or name of business rule) must not exist.

Flow of events

1. The System Administrator commands the system to initiate the creation of new entity, or selects the entity to be edited and communicates that it should be edited.
2. The system presents a form with an appropriate GUI widget for each property of the entity. There might be required information if specified by a business rule. If an entity is edited, the current value of each property is already apparent from the widget used to edit it.
3. The User enters the required information and commands the system to save the changes which completes the creation/editing of the entity, thus terminating this use case instance.

Alternative flow

In 3, if a business rule prohibits the setting of a property to the value intended by the User, the option is given for the User to change this before repeating the last command given.

Postcondition

An entity has been created/edited, or this was cancelled at some point by the User.

Create/Edit Account

Precondition

Inherited. Equivalence for accounts are on identity of the user.

Flow of events

All steps inherited.

Alternative flows

In 3, if the login is already taken, the option is given for the System Administrator to change this.

Postcondition

Inherited.

Create/Edit Business Rule

Precondition
Inherited. Equivalence on business rules are by their name and creator.

Flow of events
1. inherited from parent.
2. inherited from parent. As a minimum, a specification of the affected users should be given, as well as the action with which the rule is concerned.
3. inherited from parent.

Alternative flows
   In 3, if the ⟨rule-name, login-of-creator⟩ is already taken, the option is given for the Manager to change the rule-name.

Postcondition
   inherited.

Delete Entity, Delete Account, Delete Business Rule
These use cases has not been described further, since they are considered small, reasonably trivial, and straightforward to implement once the functionality of the Create/Edit Entity and it’s child-use cases is in place. This is not to say they are unimportant, only that their complexity is not believed to justify the more thorough treatment given to other use cases. (This is to some degree true of the Login/Logout use cases as well, but these are more often referred to.)

Login
Precondition
   The User has already had an account created with an associated login and password.

Flow of events
1. The User enters login and password, and commands the system to log on.
2. The User is logged into the system.

Alternative Flow
   In 2, if the password or login is not correct, the User is asked to retry.

Postcondition
   The User has been logged on to the system, or the provided login and password was not correct.

Logout
Precondition
   The User is logged in.

Flow of events
1. The User commands the system to log him or her out.
2. The System asks the user to confirm the intend to log off the system.

Postcondition
   The User is no longer logged in, or the user did not confirm the his or her intend to log off, and is still logged in.
Schedule Meeting

Precondition
A meeting is needed, and it has been decided whose participation should be requested.

Flow of events
1. The User indicates that a new meeting should be created and scheduled.
2. The User selects the participants.
3. A time interval is given by the User, or a specific date is suggested.
4. Based on the information provided in previous step, the system determines a number of possible dates, or the possible dates closest to the suggested date. By a possible date is meant the date full filling a criteria. The default criteria for meetings is that the all participants should be able to attend.
5. The User selects a date, and commands the system to send out a message notifying the participants of the meeting.

Postcondition
A meeting has been scheduled, that is, a representation of it is contained in the system, and it has been assigned a fixed date and time.

Arrange Board Meeting

Precondition
The board meeting has already been scheduled.

Flow of events
1. The Secretary locates the meeting, and indicates that a 'call for agenda points' action should be carried out. Note that this can be done only once for each meeting.

2. The Secretary may then optionally change the default wording of the message used for the request.
3. The secretary indicates that the message is ready and should be send.

Postcondition
Every official participating in the meeting has received a request for submission of points for the agenda.

Reserve Resource

Precondition
The User has adequate rights to reserve the intended resource.

Flow of events
1. The User selects a resource and a date and time.
2. The system responds whether the resource is available on the given date and time.
3. If the resource is available, the User may command the system to reserve it in his name.
Requirements

Alternative flow
In step 2, if the resource is not available on the given time, the User is given the opportunity to select a different date and time or a resource that can be used as a replacement (possible suggested by the system).
Also, in step 2, if the resource is not available because it is reserved by another User with lower priority, the User that initiated the current use case instance is given the opportunity to override the previous reservation.

Postcondition
The resource has been booked by the User, or it was booked by another User with higher priority or a higher prioritized task, or the current User overrode the previous reservation.

Submit Agenda Points

Precondition
Before this use case can be initiated, a meeting must be arranged, and the User in question must have received a request to submit points for the agenda.

Flow of events
1. The User decides on what points to submit, and commands the system to bring forth a form in which the points can be entered.
2. The User commands the system to submit the points.

Alternative flows
In 1, if the official has no points to submit, this is indicated, and the form is submitted.

Postcondition
The User has either submitted some points for the agenda of the meeting, or a message has been send indicating that the user in question does not have anything to put on the agenda.
2.0 Analysis

The analysis workflow is done conforming with chapter 8 in [1].

2.1 Architectural Analysis

This section will describe the identified analysis packages, obvious entity classes, and common special requirements. The work is carried out as described in section 8.6.1 in [1].

2.1.1 Analysis Packages

Since the handling of accounts is a recurring problem in administrative systems, this functionality is a candidate for constituting a separate package. The functionality for Business Rules is a candidate by the same arguments. Further, the handling of business rules and of accounts is specified by use cases having flow of events in common—hence the generalized abstract use cases Create/Edit Entity and Delete Entity.

An aspect of the system that seems to be separate from the above is the handling of meetings and resources. This functionality is separate due to the specifying use cases being applied by different actors, and due to the difference between the way meetings and accounts are applied.

The initial suggestion for division into analysis packages is therefore that there should be one analysis package for functionality pertaining to Meetings and Resources and another package for that pertaining to Accounts and Business Rules. The packages are discussed further in Section 2.5 on page 27. The distribution of use cases (really, the functionality specified by them) onto these packages are shown in figure 4.

**FIGURE 4.** Initial distribution of use case specified functionality onto analysis packages
The package for handling Accounts/Business rules is in the application general layer, while that with handling of Meetings/Resources is application specific, see p 201-202 in [1]. The Meetings/Resources package is thus dependent on the Accounts/Resources package (package dependencies not depicted).

2.1.2 Obvious entity classes

Based on the business entities of section 1, and on a glance at the use cases, the following entity classes has been identified:

**Business Rule**
A business rule is used by the system to represent just that. It is in other words a sort of policy object that holds the information of whether a given set of persons can, cannot, or must engage in some action.

**Account**
An account is just like an account in an operating system, meaning that it holds a login, password, usergroup, and other information about it’s owner that the system needs (what exactly this is will be identified in the use case realizations--analysis).

**Resource**
Resources are things such as projectors and meeting rooms that the system will manage.

**UserGroup**
A UserGroup is used in the system to designate the kind of actor that an account belongs to. UserGroups may also exist that are representative not of an actor type but of a group of users that are affected by certain business rules.

**Action**
A business rule establishes the relation between a UserGroup and an Action, see the description of BusinessRule.

**Meeting**
A meeting represents the corresponding business entity, and has as properties at least the date and time of this.
The associations between the entity classes in this initial set of analysis classes are shown on figure 5.

**FIGURE 5.** The relations among the obvious entity (analysis) classes

![Diagram showing relationships between BusinessRule, UserGroup, Account, Action, Resource, and Meeting classes]

### 2.1.3 Common Special Requirements

The special requirements are these, ordered by what they pertain to:

**Persistence**

Data such as Meetings, BusinessRules, Accounts, UserGroups and Actions should persist between sessions of use. Deletion of these entities may be done either explicitly by a User with appropriate access rights, or through automatic scheduling as specified by business rules.

**Distribution and Concurrency**

The system should be usable by many officials at the same time, and accessible through the internet.

**Security**

The use of the system should be done either from a node directly attached to the network at the Maersk Institute, or from nodes outside this network through an encrypted connection (e.g. SSL).

**Fault Tolerance**

Failure in a user node should leave the rest of the system unaffected. Failure of a more central unit (e.g. server) may cause the system to be unavailable for shorter periods until backup servers take over. Reliability of the system is important since it, once taken into use, will be a vital tool for administration of the institute.

**Transaction Management**

All modification of persistent data should be performed as atomic transactions.

### 2.2 Analysis of Use Cases

In the following sections each use case is analyzed to identify the analysis classes needed to realize it. Through diagrams it is shown how these classes collaborate in the use case realizations. Pertaining to each use case it is noted what special requirements are imposed on the system.
2.2.1 Schedule Meeting

Recall that schedule meeting is used by a User to find the date and time for a meeting based on the calendars of the participants.

Participating Analysis Classes

A user interface is needed for the User, UserUI. To decide which participants are available at a given time, it is necessary for them to have a Calendar associated with each Account. Accounts are stored in the AccountCollection. The comparison of calendars should be done by a control class, Scheduler. Finally the acceptance of a given date based on the availability should be guided by a business rule, ScheduleCriteria. What the participants participate in is a Meeting, and these are stored in the MeetingCollection. These classes and the associations among them are shown in figure 6.
Analysis Object Interactions
This describes how the analysis objects identified above are used to realize the flow of events in the use case. The description of this flow is relative to the collaboration diagram in figure 7.

**Flow of events**
The user commands the system to create a new meeting (1) which is done by the UserUI (2). Next, the User specifies who the desired participants are (3), and these are found in the AccountCollection(4). The User then proceeds to give an interval in which the meeting must be held (5). The information given by the user thus far is given to the Scheduler (6). After retrieving the calendars for the participants through their accounts (7,8) the Scheduler computes a list of possible dates/times for the meeting and suggests this to the user through the UserUI (9). The User then selects the most suitable date and time (10) and confirms the scheduling. Finally the Scheduler adds the meeting to the MeetingCollection (11).

**Special Requirements**
As described in Section 2.1.3 on page 16 the MeetingCollection and AccountCollection should be persistent and accessed only through atomic transactions.

### 2.2.2 Arrange Board Meeting
This was used by the Secretary to request points for the agenda of a previously arranged board meeting.

**Participating Analysis Classes**
A user interface is needed for the Secretary, SecretaryUI. The meeting to arrange is selected from among the board meetings in the MeetingCollection. Sending out the request is done by the AgendaCaller which gets the participants Accounts from the

---

**FIGURE 7.** Collaboration diagram for the use case realization--analysis of Schedule Meeting
Analysis

Applying the Unified Process in the design of the Maersk Institute Administrative Meeting that is selected by the secretary. These classes and their associations is shown in the class diagram, figure 8.

**FIGURE 8.**
Classes participating in the *Arrange Board Meeting* use case realization--analysis

**FIGURE 9.**
Collaboration diagram for the use case realization--analysis of *Arrange Board Meeting*

**Analysis Object Interactions**

This describes how the analysis objects identified above are used to realize the flow of events in the use case. The description of this flow is relative to the collaboration diagram in figure 9.

The Secretary browses the board meetings that have been arranged (1,2), and selects the one that should be arranged (3, 4). Next the Secretary edits the default request message send out, if needed, and commands the system to send out the message (5,6). The AgendaCaller then retrieves the accounts for the participants (7) and uses these to send the message to each participant (8).
### 2.2.3 Submit Agenda Points

This is used by a User to reply to a message requesting points for the agenda of some meeting where the user is participating.

#### Participating Analysis Classes

The User accesses the system through a UserUI. The message containing the agenda points are send using a MessageBroker. These classes and their associations are shown on figure 10.

#### Analysis Object Interactions

This describes how the analysis objects identified above are used to realize the flow of events in the use case. The description of this flow is relative to the collaboration diagram in figure 11.

The User browses the set of incoming messages, and selects the one requesting points for the agenda of some meeting (1). The user fills out the form, or indicates there is nothing to submit, and commands the system to send the reply-form. (2). The UserUI passes the message to the MessageBroker that sends it to the secretary (3).

#### Special Requirements

The secretary should receive the message within 5 minutes if s/he is logged into the system.

### 2.2.4 Reserve Resource

This was used by a User to reserve a resource for a given time and date.
Participating Analysis Classes
The User accesses the system through a UserUI. The time and date of the reservation of the Resource is entered in the Calendar. That the possible overriding of a reservation is done only according to BusinessRule is handled by the ReservationController. The resource to be reserved is selected from the ResourceCollection. These classes and their associations are shown on figure 12.

Analysis Object Interactions

(next page)
Analysis

This describes how the analysis objects identified above are used to realize the flow of events in the use case. The description of this flow is relative to the collaboration diagram in figure 13.

**FIGURE 13.**
Collaboration diagram for the use case realization--analysis of Reserve Resource

The User browses the resources in the ResourceCollection(1,2). Next, a resource is selected (3,4) as well as a date and time (5). The ReservationController then gets the calendar from the resource in question (7) and marks the resource as being reserved at the given date and time (8). Finally the reservation is confirmed to the User.

**Alternative Flow**
If the resource cannot be reserved, the flow is restarted from, and including, step 5.

**Special Requirements**
The Calendar and Resource should be persistent.

2.2.5 **Create/Edit Business Rule**
This is used by the Manager to implement a new business rule in the system.

**Participating classes**
The Manager accesses the system through the ManagerUI and uses this to create a new BusinessRule. These are kept in the BusinessRuleCollection, but before adding a rule, to this it must be validated to ensure it is not in conflict with existing rules.

**TABLE 1.**
The classes participating in the use case realization--analysis of Create/Edit BR

```
                  0..*
                  BusinessRule
  1
ManagerUI

  BusinessRuleCollection

  BusinessRuleValidator
```

**FIGURE 14.**
Collaboration diagram for the use case--realization analysis of *Create/Edit BR*

The Manager uses the ManagerUI to initiate the creation of a new business rule (1), hereby having the system create a new BusinessRule (2). The Manager then enters the properties of the business rule (3) and these are transferred to the newly created object (4). Finally the Manager confirms the creation (5) and the BusinessRuleValidator is
commanded to add the rule to the collection of Business Rules (6). The BRValidator then validates the rule (to see if it is in conflict with existing rules) (7), and if not adds it to the collection (8), and confirms the addition (9) to the Manager.

2.3 **Remaining Use Cases**

It has been chosen not to devote more attention to the remaining use cases, as they are not primary (the reason for making the system, e.g. no system is made just to be able to log in to it).

2.4 **Analysis of Classes**

This is done as advised in section 8.6.3 in [1], except at one point. Two examples in 8.6.3.1 keeps separate the descriptions of roles and responsibilities. These descriptions has been merged in this documentation, because this will make it more apparent wherefrom (e.g. what role/use case) a given responsibility comes.

Below follows a description of the roles and responsibilities for each class.

**Account**

The Account class has participating instances in the following use case: Arrange Board Meeting, where it will facilitate sending a message to it's owner. Schedule Meeting, where it will allow access to the calendar of it's owner.

**AccountCollection**

The AccountCollection is where the Accounts are stored. It is created upon installation of the system. It is used in Schedule Meeting, where it must allow selection of accounts by a some criterion. It must also allow the storing and extraction of accounts by in the use cases Delete Account, and Create/Edit Account.

**Action**

An action is created when it is needed in a business rule, and no actions exist that has the same semantics. It is used in Create/Edit Business Rule where it must hold a description of what it's semantic meaning is, as well as a meaningful name. That way it describes what is allowed or disallowed by the BusinessRule.

**AgendaCaller**

The AgendaCaller is created and lives through the execution of an instance of Arrange Board Meeting. It is used to send out messages to all participants given a Message and a Meeting.

**BusinessRule**

A BusinessRule is created by a Manager in the Create/Edit Business Rule use case. It is used in the following use cases: Create/Edit Business Rules, where it must have it's properties set. Reserve Resource, where it will indicate whether a user may reserve a given Resource, and when a Resource is already reserved, whether the reservation may be overridden by a given User. Schedule Meeting, where an instance of it will decide, given a meeting and time & date, whether the date is to be suggested as possible.

**BusinessRuleCollection**

The BusinessRuleCollection is where the BusinessRules are stored. It is created upon installation of the system. It is used in Create/Edit BusinessRule, where it must
allow selection of BusinessRules by a some criterion. It must also allow the storing and extraction of BusinessRules.

Calendar
A Calendar is created together with an Account/Resource. It is used in Reserve Resource, keep track of when a given resource is reserved. In Schedule Meeting it is used to keep track of when a user is occupied.

ManagerUI
The ManagerUI is created upon startup of the client side program of the system, when the person loggin in is a Manager. It participates in the same use cases as the Manager, that is, Create/Edit Business Rules. In this use case the Manager must be able to browse the collection of business rules that currently exist, to add new business rules and to edit (and remove) existing ones.

Meeting
A Meeting is created in Schedule Meeting. It is used in the use case Arrange Board Meeting, where it is the subject of the request-message send out, and where it must keep track of who the participants are.

MeetingCollection
The MeetingCollection is where the Meetings are stored. It is created upon installation of the system. It is used in Arrange Board Meeting, where it must allow selection of Meetings by a some criterion (e.g. all Board Meetings). It must also allow the storing and extraction of Meetings, as when a Meeting is added to it during Schedule Meeting.

MessageBroker
The MessageBroker is created and lives through the execution of an instance of Submit Agenda Points. In this use case, it is used to send the message containing the reply form to the secretary who requested the agenda points.

ReservationController
The ReservationController is created and lives through the execution of an instance of Reserve Resource. In this use case it has the responsibility to mark a resource as reserved, and to resolve any conflicts with respect to existing reservations.

Resource
A Resource is created ? (see Section 3.0 on page 27). It is used in Reserve Resource to contain relevant information about it’s real life counterpart. It must have a calendar associated with it.

ResourceCollection
The ResourceCollection is where the Resources are stored. It is created upon installation of the system. It is used in Reserve Resource, where it must allow selection of Resources by a some criterion (e.g. all Projectors or all Meeting Rooms). It must also allow the storing and retrieval of Resources, as when a Resource is added to it during Reserve Resource.

Scheduler
The Scheduler is created and lives through the execution of an instance of Schedule Meeting. It is used in this use case to find suggestions for proper time & date of a meeting, given its participants, an interval for it’s placement and some business rules.
implementing the criteria for 'proper' based on the participants and the nature of the meeting.

**SecretaryUI**

The SecretaryUI is created upon startup of the client side program of the system. It participates in the same use cases as the Secretary, that is *Arrange Board Meeting*, where it must allow the Secretary to browse all Board Meetings, select one and give a form for requesting points for the agenda from the participants of the selected Board Meeting.

**UserGroup**

A UserGroup is created when it is needed in a business rule, and no UserGroups exist that has the same semantics. Note that two UserGroups may be different even if they contain the same Users. It is used in *Create/Edit Business Rule* where it must hold a description of what it's semantic meaning is, as well as a meaningful name. That way it describes what is allowed or disallowed by the BusinessRule.

**UserUI**

The UserUI is created upon startup of the client side program of the system. It participates in the same use cases as the User, these are: *Schedule Meeting*, where it must allow the User to create a meeting, give a form for interval, selection of participants and acceptance of the time and date suggested by the system. *Submit Agenda Points*, where the User must be able to browse messages, select one containing a request for agenda-points, and fill out and send a standard reply form. *Reserve Resource*, where the User should browse and select resources, fill out a reservation form, and submit the reservation.

**BusinessRuleValidator**

The BusinessRuleValidator is created in the Create/Edit Business Rule use case. It is used within the realization of this use case to validate that a newly created business rule does not conflict with those already existing in the BusinessRuleCollection.

**Associations and aggregations**

The associations are shown in figure ???. Apart from the Collection classes containing instances of the classes to which they are dedicated, the controll classes *AgendaCaller* and *MessageBroker* are related because the former needs to be able to send messages. The precise nature of this relationship will be decided upon during design, as it may be any of aggregation, generalization, and association. (the real issue here is how best to reuse the code in the *MessageBroker*, and that depends on how this class will be shaped in design.)

**Generalizations**

The Collection classes clearly share a lot, but what they share is a part of standard api’s in most programming languages, so the whether to express this through a general class or a parameterized one is better postponed to design.

Another striking commonality between most of the entity classes (account, meeting, business rule, action, resource ...) is the way they should be handled by the interface classes. The precise nature of this is as well believed to be better dealt with in design.
2.5 Analysis of Packages

Relative to the two packages identified in Section 2.1.1 on page 14, and shown on figure 4, the distribution of the analysis classes onto these packages are shown in figure 15. The classes with a fill colour are those in the application specific package, whereas those that are white are in the application general package. Note that the only purpose of this diagram is to visualize the class-level dependency among the two packages. What can be noted about this is that the application general package is dependent on the application specific one in only the case of Account having to know of Calendar. This problem will be dealt with in design, to make the application general package totally independent of the application specific one.

3.0 Missing Features discovered in First Iteration

In performing the first iteration, a number of shortcomings were detected.

1. How a message gets to a user, browsing of messages by a user etc.
2. That a user should be able to send a ‘free-form’ message to another.
3. It should be possible to create/delete/edit Resources.
4. It should be possible for a User to edit his or her calendar.
5. Deletion/Editing of meetings.
6. During analysis it has become clear that there are many similarities in the handling of Resources and Entities (Business Rules, Accounts). This, held together with item 3 above, suggests the need for a correction of the use case model.
4.0 Design

The design follows the chapter 9 in [1].

4.1 Architectural Design

This is according to 9.5.1 in [1].

Nodes and Network Connections

Three network nodes are needed in the system, plus backup of these. They are arranged in a standard three-tier architecture. The nodes are shown in figure 16.

FIGURE 16. Nodes and Network Connections

Nodes

- A server to implement business logic
- A DBMS server to store persistent data
- A reasonably thin client for each user of the system.

Connections

Any connection to the client is over either the internet using vpn, or within the intranet. This connection cannot, therefore, be assumed to be more stable than the average internet connection. With respect to security, it lies in the principle of vpn, that the security of the line is assumed to be equal to that of an intranet connection.

Connections between the Business Logic node and the DBMS server is through the intranet. This means that a higher bandwidth is expected for this connection.

Backup Servers (not shown)

The persistent data such as meetings and accounts are vital for the continued work of administering business processes at the institute, so the database should be backed up. Though loosing a meeting due to (rare) server crashes is inconvenient, a little time is expendable on this account, so a backup server will do instead of a redundant DB.
4.1.1 Subsystems and their Interfaces
As in 9.5.1.2 pp. 234 in [1]

Application Subsystems
Relative to the analysis packages, the packages accounts and brules are traceable to the analysis package Accounts/Business Rules Handling. The mias (abbreviation for Maersk Institute Administrative System) package and entityeditor packages are traceable to the Meeting/Resource Handling packages. Note that functionality now located in the entityeditor package has moved from the application specific to the application general layer. This is because it is evident from the analysis model (specifically the use case realizations--analysis), that entities such as Resource, Meeting, Account, Business Rule all have in common that they should be created/edited/deleted through the various user interfaces. This commonality might as well be extracted for several reasons. Firstly it will help create a more uniform userinterface, secondly it will remove some redundancy in gui classes, thirdly it will make it easier to add gui support for new entities, an finally the resulting subsystem may be reused across applications.

Middleware/System-Software Subsystems
Java will be used as implementation language, so the packages defined in this api provides a natural base for building the application subsystems.

Further, an RDBMS will be used, but the peculiarities of this are handled at a lower level, namely the jdbc api.

Subsystem Dependencies
Shown in figure 17. Note that there is only a dependency directed from the mias package towards the accounts package, not the reverse as is implied by the bidirectional association between Calendar and Account in the diagram, Figure 15 on page 27. It is imagined that this reference is kept in a decorator class contained within the mias package, in order to make the accounts package totally independent of the mias package.
Subsystem Interfaces

From the class diagram showing distribution of analysis classes into analysis packages, Figure 15 on page 27, the interfaces that apply to the dependencies can be specified further.

The dependency of mias on accounts is really a dependency of calendars needing to know it’s owner, and of controll classes that should be able to extract information from accounts, so the interface is called UserAccount.

The dependency of mias on brules is due to the controll classes in mias needing to incorporate business rules into their behaviour, so the interface is called BusinessRule.

An interface for entityeditor cannot be decided upon at this time since it will be dependent upon how it is considered best to incorporate the entityeditor into the gui classes. This will be decided when the gui classes are more apparent.

4.2 Architecturally Significant Design Classes

5.0 References

Applying the Unified Process in the design of the Maersk Institute Pervasive Administrative System

Mads Ingstrup

This report contains together with a model in Rational Rose the documentation of the results of applying the Unified process to the development of the pervasive system for the Maersk Institute. It thus documents activity two in the array of activities described in the msc report.
Table of Contents

1.0 Requirements 3
  1.1 Introduction 3
  1.2 Business Model 3
    1.2.1 Business Actors 3
    1.2.2 Business Use Cases 3
    1.2.3 Business Object Model 6
  1.3 Use Case Model 6
    1.3.1 Actors 6
    1.3.2 Use Cases -- Brief Description 7
    1.3.3 The use case model as a whole 8
    1.3.4 Architecture Description/Prioritizing the Use Cases 9
    1.3.5 Detailed Description of each Use Case 9

2.0 Analysis 14
  2.1 Architectural Analysis 14
    2.1.1 Analysis Packages 14
    2.1.2 Obvious entity classes 15
    2.1.3 Common Special Requirements 17
  2.2 Analysis of Use Cases 17
    2.2.1 Prepare Presentation 17
    2.2.2 Conduct Board Meeting 19
  2.3 Remaining Use Cases 21
  2.4 Analysis of Packages 22

3.0 Design 24
  3.1 Architectural Design 24
    3.1.1 Subsystems and their Interfaces 25
  3.2 Architecturally Significant Design Classes 26

4.0 Glossary of terms 26

5.0 References 26
1.0 Requirements

This is documents the workflow of gathering requirements cf. the workflow described in section 7.4 in [1]. Comments about how this document relates to the documentation report of Mitas are set in a smaller sans serif font and enclosed in parantheses, for instance (~ some comment ~).

1.1 Introduction

The system that is to be developed is a pervasive system for use at the Maersk McKinney Møller Institute for Production Technology. It is build on top of the Mitas, and the extensions will be in making the system become pervasive, hence the name Maersk Institute Pervasive Administrative System, Mipas. The primary purpose will be:

- To keep track of the persons that are at the institute.
- Considering the use cases of Mitas, at least Reserve Resource and Submit Agenda Points should be possible to perform from a mobile unit, such as a PDA/mobile phone.
- In Mitas, arrangement of meetings were supported, e.g. those activities taking place before the actual meeting is conducted. Because Mipas is pervasive it becomes possible to provide support during the actual meeting. (In fact, the “pervasive meeting room” is an often used example of pervasive computing, see for instance [cite{pervasive-meeting-research-mitas}])
- To provide support for other routine activities that can not be adequately supported by traditional desktop applications (use of various artefacts e.g. projectors, automating entering of events into documents)

1.2 Business Model

The business model is largely the same, because nothing about the institute has changed since the partial development of Mitas. Only business actors and use cases that are additions to the Business Model of Mitas will be described.

1.2.1 Business Actors

The Secretary and Official are included in this BM as well. The following business actors also appear in the new business use cases

User
A person who visits the institute or somehow uses it in another way, without holding some official position there. Examples include representatives for corporate project partners, other institutes, the dean etc.

1.2.2 Business Use Cases

The following additional business use cases are considered relevant for the system.

- Prepare Presentation
- Conduct Board Meeting
The Business Use Case Model as a Whole

This description includes all use cases in the Mipas system, in order to present it in its entirety.

The Secretary may use the Arrange Board Meeting use case to call for points to the agenda of the meeting, when a board meeting has already been scheduled using Schedule Meeting. The Arrange Board Meeting requires the assistance of several Officials, who are the participants in the meeting and thus has the right to put points on the agenda. Conduct Board Meeting is used in performing the actual meeting. The Prepare Presentation is used by an Official to make sure that the arrangements with respect to tools used in the presentation are in place.

Turning to the use cases that can be initiated by an official, the Schedule Meeting is used to find a date and time for a meeting when it’s participants are known. The Reserve Projector and Reserve Meeting Room are used by an Official to reserve a resource, for a given period of time at a given date and time.

Below follows a description of those of the business use cases that are additional to MIPAS. Each is described in terms of flow of events and pre- and postconditions.

Prepare Presentation

Precondition

The Official has been scheduled for performing a presentation at an already arranged meeting.

Flow of events

1. At some point the Official creates some powerpoint-like slides to be used during session x.
Requirements

2. At some point the Official creates some powerpoint-like slides to be used during session x.
3. The Official then checks whether s/he needs to bring a projector, or whether some other participating Official already has planned to bring one.
4. If the Official owns a laptop, the slides are transferred to this and it is brought to the meeting. Otherwise, the presentation is send to another participating Official, who will bring a laptop, and upon agreement let the first Official use this to present from.

Auxiliary Requirements

If the user relies on the availability of a projector brought by some other participating Official, and this participant cancels his or her participation in the meeting, the first Official is told, and will need to re-perform the last step.

Postcondition

The Official has prepared a presentation and arranged that the required machinery is available to support it at the meeting in question.

Conduct Board Meeting

Precondition

A meeting has been arranged, and it is scheduled for the current time and date.

Flow of events

1. The secretary notes the time of arrival for each delayed participant, and the time the meeting started.
2. For each point on the agenda, as the point is discussed, the secretary takes notes that will later constitute the content of the minutes for this point of the agenda.
3. If a participant leaves before the meeting ends, this is noted by the secretary and later added to the preample of the minutes.
4. The secretary notes if any points are postponed to the next meeting, and writes down the time the meeting stopped.
5. Later, when the secretary has written the formal document containing the minutes of the meeting, based on the notes made, these minutes are send by email to every participant.

Postcondition

The minutes has been send to each participant, or too few of the scheduled participants showed up, and the entire meeting must be rescheduled.
1.2.3 Business Object Model

Below a diagram of the business entities and their relations are shown.

The Calendar is a physical book into which reservation entries are entered. The Projector is one of the resources that may be reserved by entering an entry in its Calendar, as is the Meeting Room. A Meeting is associated with precisely one Meeting Room, which is the place where it is held. A Meeting Room may of course be associated with many Meetings, as long as they are scheduled for different times, a constraint that is ensured by the Calendar for the Meeting Room. A Board Meeting is a special kind of meeting that is more formal, and always results in precisely one instance of Meeting Minutes, for documenting the meeting. Presentations may be used a all kinds of meetings. Both Meeting Minutes and Presentations are a kind of Documents. Note that the new classes are the two kinds of documents, because the creation and use of these two types of documents are a target for a pervasive computing application.

(~ following paragraph unchanged from last report ~)

A Business Rule is less tangible than are the other entities. It represents some constraint on what can happen in the business. A business rule may be written down as a formal rule, or it may be more tacit, if there is a common but unarticulated agreement about it. For instance a reservation for a meeting room may be overridden in some cases, such as when some students has reserved a meeting room for watching a movie, and it later turns out that an exam should take place in the room at the same time. A business rule will in this context be characterized by the persons, groups of persons, or entities that it is concerned with, and how these relate to some action.

1.3 Use Case Model

1.3.1 Actors

The following actors occur in the use cases for MIPAS.
Requirements

Official
The Official actor in the use case model can be traced back to the Official actor in the Business Model. The role of Official is thus played by everyone who plays an active role at the institute, ranging from tasks such as participating in meetings to using resources at the institute.

User
The User is different from the actor by the same name in the use case model for Mitas. This role is introduced in recognition of the fact that not everyone that comes to the institute hold an official position there, and thus cannot be expected to have a Mitas account. This actor can be traced back to the business actor User.

(~ Mitas specific actors left out ~)

1.3.2 Use Cases -- Brief Description
Currently, these pervasive use cases has been identified:

Conduct Board Meeting
is used by a Secretary whenever a board meeting has been scheduled, and the meeting is about to take place.

Prepare Presentation
is used by a User to, once a presentation is made in some application, associate this document with a given meeting and ensure that the means will be available to perform the presentation.

Check In/Out
This is used by a User to notify Mitas that s/he has arrived/departed from the institute.

Locate Entity
Is used by a User to find the location of some entity, be it a projector, a person or even a room if the user is new to the institute.

Send/Receive Message
is used by a User to communicate with other users. It is imagined as a sort of extension of instant messaging.

Edit Visibility Status
is used by a User to control how much others are allowed to be aware of his or her presence. It is imagined that this includes current location, plan for the day, whether present at all; status in the form of busy, available, on the phone etc.

A number of Mitas use cases should be accessible through other channels. These are

Submit Agenda Points
is used by a User to submit points for an agenda for a particular meeting. It may be accessed either when logged into a stationary or through a PDA.

Reserve Resource
is used by a User to reserve a resource for a given period of time at a given time and place. May be accessed from stationary/PDA.
1.3.3 The use case model as a whole

In Figure 3 on page 8 a use case diagram shows all the use cases and actors and the relations among these. *Locate Meeting Room* and *Locate Projector* are only shown as illustration of how *Locate Entity* is derived from the business use cases; they seem similar enough that any distinction is superfluous (from a usage perspective).

**FIGURE 3.** Use Case Diagram

Starting with the Check In/Out use cases, a user may use these to (un)register his or her presence in the system. Using PDAs the Users may communicate by sending messages using the Send/Receive Message Use cases. At any time Users may use Edit Visibility Status to indicate how much of their presence is visible to other users. For instance it may affect whether a user can be located by an Official through the Locate Entity use case, which may also be used to locate physical entities such as projectors. Officials may employ the Conduct Meeting use case to receive aid in documenting a board-like meeting. The other major kind of meeting activity is presentations. They are supported in the planning phase where an Official may use Prepare Presentation to associate a presentation with a given meeting, and optionally use Reserve Resource to ensure the presence of a capable projector for the presentation.
1.3.4 **Architecture Description/Prioritizing the Use Cases**

The prioritization has been carried out as advised in 12.6 in [1].

First keep in mind that the Mipas system is an extension of Mitas. That there are no thoroughly described stable architecture for Mitas is no objection to prioritizing the use cases for Mipas, since the criteria are the same. The criteria are that the two primary risks listed in 12.6 in [1] should be mitigated as early in the development cycle as possible. They are (1) the risk of not getting the architecture right and (2) the risk of not getting requirements right. The second of these is not relevant in this case, since the project is made by one person who thus plays both the role of user and developer. The risk of not getting the architecture right means prioritizing the architecturally significant use cases, which are those "that cover the main tasks or functions the system is to accomplish." [1] p.331.

The reason for building Mipas is to exploit the possibilities afforded by pervasiveness. This means that the use cases Apply Projector, Prepare Presentation, Conduct Meeting becomes important. From a user perspective the privacy model specified through the Edit Visibility Status is important in getting the users to use the system.

The prioritizations thus become:

*Primary Use Cases:*
- Conduct Meeting, Prepare Presentation, Apply Projector, Edit Visibility Status

*Secondary Use Cases:*
- Reserve Resource, Locate Entity, Check in/out, Submit Agenda Points

*Ancillary*
- Read/Send Message

1.3.5 **Detailed Description of each Use Case**

Each of the descriptions given below follows the form already applied in the descriptions of the business use cases. Since it is the pervasive aspect of Mipas that is in focus for the project, only a subset of the use cases appearing on the above diagram is described thoroughly.

**Conduct Board Meeting**

*Precondition:*

A meeting has been scheduled at the current time and place of the initiating Official.  
[How to specify this constraint of current place for a use case ?]

*Flow of events*

1. The initiating Official (effectively the secretary for the meeting) commands the system to start the note-taking function and the time, date and participants are entered into the document containing the minutes of the current meeting.
2. Going through the points on the agenda, whenever one has been discussed the official enters the results of the discussion into the minutes.
3. Whenever a user enters or leaves the room, the time of arrival and id of the user is entered into the minutes.
4. At the end of the meeting the Official exits the room thus making the system file the minutes in a place predefined for these kinds of meetings.
Post condition
A document has been created, containing the minutes of the meetings.

Prepare Presentation

Precondition
The Official has an entry for session x in his calendar.

Flow of events
1. At some point the user creates some powerpoint-like slides to be used during session x, and makes a link from the calendar entry for x to the slideshow.
2. The system then checks whether a projector will be available during session x, and that it is capable of displaying slides. If no resource has been reserved, either by the user or some other participant the system suggests that the user reserves a capable projector. (<inclusion point for Reserve Resource>)
3. If the standard equipment carried around by the Official (e.g. a PDA) is not advanced enough to communicate with the projector, the system notifies the user that something more advanced must be brought, and possibly comes up with a suggestion based on knowledge about what is accessible to this user.

Auxiliary Requirements
If the user relies on the availability of a projector reserved by some other participant of the meeting, and this participant cancels the reservation of the projector, or cancels his or her participation in the meeting, the Official must be notified of this change.

Postcondition
The Official has prepared a presentation and associated it with a particular meeting.

Locate Entity

Precondition
If it is a person that must be located he or she must already be registered as an official [user] of the system.

Flow of events
1. The Official identifies the person or resource to be located. This may be done by browsing for the person, the resourcetype etc. or by entering an id
2. If it is a User and the visibility settings chosen by this User permits so, the system presents the location of the official.

Post Condition
The Official has found the entity that was searched for, or this could not be found an the Official was notified hereof (includes an official whose privacy settings does not allow Mipas to reveal his or her location).

Check In

Precondition
The User is known to the system, i.e. has an account.
Requirements

Flow of events
1. The user enters the lobby of the institute, and moves to the Doorman Agent.
2. The User identifies him- or herself, if the institute was not entered outside normal opening hours, in which case the identity is already known through the users key-card.
3. Optionally the user specifies a visibility different from his default setting, and confirms the checkin. <extension point for Edit Visibility Status>
4. If allowed by the visibility settings, the system notifies other users that have registered to know when the person in question arrives.

Postcondition
The User is now registered as being present at the institute.

Check Out
Precondition
The User is located at the entry of the institute (or some other exit).

Flow of events
1. The User commands the system to log him out.
2. If allowed by the visibility settings, the system notifies other users that have registered to know when the person in question leaves.

Postcondition
The User is now registered as not being present at the institute.

Apply Projector
Precondition
The Projector has been reserved for the current time, by the Official attempting to use it, and it is colocated with him/her. (If it has not been booked, then this should be done first in order to let others know who are using it).

Flow of events
1. The system connects the user, who is there and scheduled for presentation, to the projector and notifies him/her that the connection is set up. The user may then command the system to show the slides through his PDA.

Postcondition
The Official has completed the application of the resource, and indicated this to the system, through the PDA.

Auxiliary requirements
During the Official’s presentation, the projector is locked and cannot be accessed by other parties without the consent of the Official.

Nonfunctional requirements
The connection between the Officials PDA and the projector is wireless.
Requirements

Schedule Meeting

Precondition
A meeting is needed, and it has been decided whose participation should be requested.

Flow of events
1. The User indicates that a new meeting should be created and scheduled.
2. The User selects the participants.
3. A time interval is given by the User, or a specific date is suggested.
4. Based on the information provided in previous step, the system determines a number of possible dates, or the possible dates closest to the suggested date. By a possible date is meant the date full filling a criteria. The default criteria for meetings is that the all participants should be able to attend.
5. The User selects a date, and commands the system to send out a message notifying the participants of the meeting.

Postcondition
A meeting has been scheduled, that is, a representation of it is contained in the system, and it has been assigned a fixed date and time.

Arrange Board Meeting

Precondition
The board meeting has already been scheduled.

Flow of events
1. The Secretary locates the meeting, and indicates that a 'call for agenda points' action should be carried out. Note that this can be done only once for each meeting.
2. The Secretary may then optionally change the default wording of the message used for the request.
3. The secretary indicates that the message is ready and should be send.

Postcondition
Every official participating in the meeting has received a request for submission of points for the agenda.

Reserve Resource

Precondition
The User has adequate rights to reserve the intended resource.

Flow of events
1. The User selects a resource and a date and time.
2. The system responds whether the resource is available on the given date and time.
3. If the resource is available, the User may command the system to reserve it in his name.

Alternative flow
In step 2, if the resource is not available on the given time, the User is given the opportunity to select a different date and time or a resource that can be used as a replacement (possible suggested by the system). Also, in step 2, if the resource is not available because it is reserved by another User with lower priority, the User that initiated the current use case instance is given the opportunity to override the previous reservation.

**Postcondition**

The resource has been booked by the User, or it was booked by another User with higher priority or a higher prioritized task, or the current User overrode the previous reservation.

**Submit Agenda Points**

**Precondition**

Before this use case can be initiated, a meeting must be arranged, and the User in question must have received a request to submit points for the agenda.

**Flow of events**

1. The User decides on what points to submit, and commands the system to bring forth a form in which the points can be entered.
2. The User commands the system to submit the points.

**Alternative flows**

In 1, if the official has no points to submit, this is indicated, and the form is submitted.

**Postcondition**

The User has either submitted some points for the agenda of the meeting, or a message has been send indicating that the user in question does not have anything to put on the agenda.
2.0 Analysis

The analysis workflow is done conforming with chapter 8 in [1].

2.1 Architectural Analysis

This section will describe the identified analysis packages, obvious entity classes, and common special requirements. The work is carried out as described in section 8.6.1 in [1].

2.1.1 Analysis Packages

The allocation of use cases to analysis packages has been done as suggested in 8.6.1.1 in [1], based on what business processes the use cases support.

The first package will contain those classes needed to implement use cases for meeting support. This is at the application specific layer, because it is more specific to the application that the functionality in the application general layer, although it may be argued that some functionality for meeting support is generally applicable.

Turning to the application general layer, two packages have been identified. Quite a large proportion of the use cases are concerned with the management of users access, visibility, accounts etc. Use cases focusing on this area are in the User Accounts package. The other package is for the management of resources, including tracking their positions. The initial suggestion for division of use cases into analysis packages (really, the functionality specified by them) is therefore as depicted in figure 4. The packages are discussed further in Section 2.4 on page 22.
2.1.2 Obvious entity classes

Based on the business entities of section 1, and on a glance at the use cases, the following entity classes has been identified:

**RoomAgent**

is a representative of a room. It does not in itself support what happens in the room, but provides a base for that other classes can interact with to access the room’s features (projectors, window shades etc.).

**UserAgent**

is a User’s representative within the system. It is the medium used by other classes for communicating with a given user. It may also be responsible for performing tasks that a user has scheduled.

**ResourceAgent**

is the representation of a resource within the system. In general it is responsible for letting users

**Projector**

has a number of properties that characterize and identifies a projector.

---

**FIGURE 4.**

Initial distribution of use case specified functionality onto analysis packages

**Application Specific Layer**

Meeting Support

Prepare Presentation

Submit Agenda Points

**Application General Layer**

User Accounts

Resource Tracking

Check in

Check out

Apply Projector

Conduct Board Meeting

Check in

Read Message

Edit Visibility Status

Reserve Resource

Locate Entity

Prepare Presentation

<include>

Apply Projector

Conduct Board Meeting

Check in

Read Message

Edit Visibility Status

Reserve Resource

Locate Entity

Apply Projector

Conduct Board Meeting

Check in

Read Message

Edit Visibility Status

Reserve Resource

Locate Entity

Apply Projector

Conduct Board Meeting

Check in

Read Message

Edit Visibility Status

Reserve Resource

Locate Entity

Apply Projector

Conduct Board Meeting

Check in

Read Message

Edit Visibility Status

Reserve Resource

Locate Entity

Apply Projector

Conduct Board Meeting

Check in

Read Message

Edit Visibility Status

Reserve Resource

Locate Entity

Apply Projector

Conduct Board Meeting

Check in

Read Message

Edit Visibility Status

Reserve Resource

Locate Entity

Apply Projector

Conduct Board Meeting

Check in

Read Message

Edit Visibility Status

Reserve Resource

Locate Entity

Apply Projector

Conduct Board Meeting

Check in

Read Message

Edit Visibility Status

Reserve Resource

Locate Entity

Apply Projector

Conduct Board Meeting

Check in

Read Message

Edit Visibility Status

Reserve Resource

Locate Entity

Apply Projector

Conduct Board Meeting

Check in

Read Message

Edit Visibility Status

Reserve Resource

Locate Entity

Apply Projector

Conduct Board Meeting

Check in

Read Message

Edit Visibility Status

Reserve Resource

Locate Entity
Document
in general documents a meeting. There are several kinds of documents, but these are not shown to avoid too many details.

MeetingManager
is the class that will manage the ongoing support of a meeting. There is at least one for each meeting, and it often produces a document, (e.g. minutes).

Meeting
is an entity holding the time, date, participants or the real-life meeting it represents.

ResourceTracker
is used by the MeetingManager to enable it to receive notifications about who leaves/enters the meeting room, and who are at a given area (e.g. near the board or a projector).

Account
An account is just like an account in an operating system, meaning that it holds a login, password, usergroup, and other information about it’s owner that the system needs (what exactly this is will be identified in the use case realizations--analysis).

Resource
Resources are things such as projectors and meeting rooms that the system will manage.

The associations between the entity classes in this initial set of analysis classes are shown on figure 5.

**FIGURE 5.** The relations among the obvious entity (analysis) classes
2.1.3 Common Special Requirements

The special requirements are these, ordered by what they pertain to:

Persistence
Data such as Meetings, Accounts, Documents should persist between sessions of use. Deletion of these entities may be done either explicitly by an Official with appropriate access rights, or through automatic scheduling as specified by business rules.

Distribution and Concurrency
The system should be usable by many officials at the same time, and accessible through the internet.

Security
The use of the system should be done either from a node directly attached to the network at the Maersk Institute, or from nodes outside this network through an encrypted connection (e.g. SSL). All wireless connections should be encrypted.

Fault Tolerance
Failure in a user node should leave the rest of the system unaffected. Failure of a more central unit (e.g. server) may cause the system to be unavailable for shorter periods until backup servers take over. Reliability of the system is important since it, once taken into use, will be a vital tool for administration of the institute.

Transaction Management
All modification of persistent data should be performed as atomic transactions. The special requirements are these, ordered by what they pertain to.

2.2 Analysis of Use Cases

In the following sections the most important use cases are analyzed to identify the analysis classes needed to realize them. Through diagrams it is shown how these classes collaborate in the use case realizations. Pertaining to each use case it is noted what special requirements are imposed on the system.

2.2.1 Prepare Presentation
Recall that Prepare Presentation is is used by a User to, once a presentation is made in some application, associate this document with a given meeting and ensure that the means will be available to perform the presentation.

Participating analysis classes
The Official must have an interface, OfficialUI. The purpose is to associate a given Presentation, a kind of Document, with a particular Meeting. Meetings are stored in the MeetingCollection. The PresentationManager controls the flow of the use case. These analysis classes are shown in the class diagram, figure .
Analysis Object Interactions
This describes how the analysis objects identified above are used to realize the flow of events in the use case. The description of this flow is relative to the collaboration diagram in figure.

(moved to next page, to keep diagram and flow-of-events description together.)
Flow of events
The User creates a slideshow (at a stationary, and places them where accesible to the OfficialUI, UserAgent) and selects it (1). The user then browses the collection of meetings (2) and selects the one that the slideshow is supposed to be used at (3). The Official then indicates to the system that the selected slideshow should be used at the selected meeting (4). Next, the PresentationManager checks that a projector will be available in at the meeting (e.g. there is a stationary in the meeting room, or another participant in the meeting has reserved one.) (5). If this is not the case the PresentationManager indicates this to the user, and optionally starts the Reserve Resource use case (optional; not shown). The PresentationManager then checks whether the PDA that the Official in question always has is sufficient for showing Slideshows on the available projector (6). If this is not the case the PresentationManager notifies the user of this (optional; not shown).

2.2.2 Conduct Board Meeting
Recall that Conduct Board Meeting is used by a Secretary whenever a board meeting has been scheduled, and the meeting is about to take place.

Participating Analysis Classes
The Secretary uses the system through the SecretaryUI. Exactly one Meeting is managed by the MeetingManager at a time, and this one is documented in an instance of Meeting-Minutes. More than one of these may be viewed simultaneously by the Secretary, if s/he for instance needs to see what happenend at a previous meeting. These classes and their relations are shown in the class diagram, figure 7.
Analysis

**Analysis Object Interactions**

This describes how the analysis objects identified above are used to realize the flow of events in the use case. The description of this flow is relative to the collaboration diagram in figure 8.

**Flow of events**

The initiating Official (effectively the secretary for the meeting) commands the system to start the meeting and thus the note-taking function (1). Next, the MeetingManager creates a new document of type MeetingMinutes (2) and enters date, time and present participant in the preamble of the document (3). The Official [secretary] types in some notes for each point on the agenda (4). During this time, if a participant leaves the meeting prematurely, this is entered into the MeetingMinutes (5). Finally, the Official lets the system know that the meeting has ended (6), and the closing time of the meeting is entered into the Minutes (7), and the Minutes are added to the Meeting that they document (8).
2.3 Remaining Use Cases

It has been chosen not to devote more attention to the remaining use cases, as this is not believed to reveal any further problems about modelling a pervasive system.

Analysis of Classes

This is done as advised in section 8.6.3 in [1], except at one point. Two examples in 8.6.3.1 keeps separate the descriptions of roles and responsibilities. These descriptions has been merged in this documentation, because this will make it more apparent where-from (e.g. what role/use case) a given responsibility comes.

Meeting
A Meeting is created in Schedule Meeting. It is used in the use case Arrange Board Meeting, where it is the subject of the request-message send out, and where it must keep track of who the participants are.

SecretaryUI
The SecretaryUI is created upon startup of the client side program of the system. It participates in the same use cases as the Secretary, that is Arrange Board Meeting, where it must allow the Secretary to browse all Board Meetings, select one and give a form for requesting points for the agenda from the participants of the selected Board Meeting.

MeetingCollection
The MeetingCollection is where the Meetings are stored. It is created upon installation of the system. It is used in Arrange Board Meeting, where it must allow selection of Meetings by a some criterion (e.g. all Board Meetings). It must also allow the storing and extraction of Meetings, as when a Meeting is added to it during Schedule Meeting.

MessageBroker
The MessageBroker is created and lives through the execution of an instance of Submit Agenda Points. In this use case, it is used to send the message containing the reply form to the secretary who requested the agenda points.

ReservationController
The ReservationController is created and lives through the execution of an instance of Reserve Resource. In this use case it has the responsibility to mark a resource as reserved, and to resolve any conflicts with respect to existing reservations.

Resource
A Resource is created ? (see Section on page 23). It is used in Reserve Resource to contain relevant information about it's real life counterpart. It must have a calendar associated with it.

ResourceCollection
The ResourceCollection is where the Resources are stored. It is created upon installation of the system. It is used in Reserve Resource, where it must allow selection of
selection of Resources by a some criterion (e.g. all Projectors or all Meeting Rooms). It must also allow the storing and retrieval of Resources, as when a Resource is added to it during Reserve Resource.

**UserUI**

The UserUI is created upon startup of the client side program of the system. It participates in the same use cases as the User, these are: Schedule Meeting, where it must allow the User to create a meeting, give a form for interval, selection of participants and acceptance of the time and date suggested by the system. Submit Agenda Points, where the User must be able to browse messages, select one containing a request for agenda-points, and fill out and send a standard reply form. Reserve Resource, where the User should browse and select resources, fill out a reservation form, and submit the reservation.

**Associations and aggregations**

The associations are shown in figure ???. Apart from the Collection classes containing instances of the classes to which they are dedicated, the control classes AgendaCaller and MessageBroker are related because the former needs to be able to send messages. The precise nature of this relationship will be decided upon during design, as it may be any of aggregation, generalization, and association. (the real issue here is how best to reuse the code in the MessageBroker, and that depends on how this class will be shaped in design.)

**Generalizations**

The Collection classes clearly share a lot, but what they share is a part of standard api’s in most programming languages, so the whether to express this through a general class or a parameterized one is better postponed to design.

Another striking commonality between most of the entity classes (account, meeting, business rule, action, resource ...) is the way they should be handled by the interface classes. The precise nature of this is as well believed to be better dealt with in design.

**2.4 Analysis of Packages**

Relative to the two packages identified in Section 2.1.1 on page 14, and shown on figure 4, the distribution of the analysis classes onto these packages are shown in figure 9. The classes with a fill colour are those in the application specific package, whereas those
that are white are in the application general package. Note that the only purpose of this diagram is to visualize the class-level dependency among the two packages. What can be noted about this is that the application general package is dependent on the application specific one in only the case of Account having to know of Calendar. This problem will be dealt with in design, to make the application general package totally independent of the application specific one.
3.0 Design

The design follows the chapter 9 in [1].

3.1 Architectural Design

This is according to 9.5.1 in [1].

Nodes and Network Connections

Three network nodes are needed in the system, plus backup of these. They are arranged in a standard three-tier architecture. The nodes are shown in figure 10.

![Nodes and Network Connections](image)

**Nodes**
- A server to implement business logic
- A DBMS server to store persistent data
- A reasonably thin client for each user of the system.

**Connections**

Any connection to the client is over either the internet using vpn, or within the intranet. This connection cannot, therefore, be assumed to be more stable than the average internet connection. With respect to security, it lies in the principle of vpn, that the security of the line is assumed to be equal to that of an intranet connection.

Connections between the Business Logic node and the DBMS server is through the intranet. This means that a higher bandwidth is expected for this connection.

**Backup Servers (not shown)**

The persistent data such as meetings and accounts are vital for the continued work of administering business processes at the institute, so the database should be backed up. Though loosing a meeting due to (rare) server crashes is inconvenient, a little time is expendable on this account, so a backup server will do instead of a redundant DB.
3.1.1 Subsystems and their Interfaces
As in 9.5.1.2 pp. 234 in [1]

Application Subsystems
Relative to the analysis packages, the packages accounts and brules are traceable to the analysis package Accounts/Business Rules Handling. The mias (abbreviation for Maersk Institute Administrative System) package and entityeditor packages are traceable to the Meeting/Resource Handling packages. Note that functionality now located in the entityeditor package has moved from the application specific to the application general layer. This is because it is evident from the analysis model (specifically the use case realizations--analysis), that entities such as Resource, Meeting, Account, Business Rule all have in common that they should be created/edited/deleted through the various user interfaces. This commonality might as well be extracted for several reasons. Firstly it will help create a more uniform userinterface, secondly it will remove some redundancy in gui classes, thirdly it will make it easier to add gui support for new entities, an finally the resulting subsystem may be reused across applications.

Middleware/System-Software Subsystems
Java will be used as implementation language, so the packages defined in this api provides a natural base for building the application subsystems.

Further, an RDBMS will be used, but the peculiarities of this are handled at a lower level, namely the jdbc api.

Subsystem Dependencies
Shown in figure 11. Note that there is only a dependency directed from the mias package towards the accounts package, not the reverse as is implied by the bidirectional association between Calendar and Account in the diagram, Figure 9 on page 23. It is imagined that this reference is kept in a decorator class contained within the mias package, in order to make the accounts package totally independent of the mias package.

FIGURE 11. Design subsystems with interfaces and dependencies.
Subsytem Interfaces

From the class diagram showing distribution of analysis classes into analysis packages, Figure 9 on page 23, the interfaces that apply to the dependencies can be specified further.

The dependency of mias on accounts is really a dependency of calendars needing to know its owner, and of controll classes that should be able to extract information from accounts, so the interface is called UserAccount.

The dependency of mias on brules is due to the controll classes in mias needing to incorporate business rules into their behaviour, so the interface is called BusinessRule.

An interface for entityeditor cannot be decided upon at this time since it will be dependent upon how it is considered best to incorporate the entityeditor into the gui classes. This will be decided when the gui classes are more apparent.

3.2 Architecturally Significant Design Classes

...