Modelling of Context: Designing Mobile Systems from Domain-Dependent Models

Nielsen, Peter Axel; Stage, Jan

Published in:
Proceedings of 32nd Information Systems Research Seminar in Scandinavia

Publication date:
2009

Document Version
Publisher's PDF, also known as Version of record

Modelling of Context: Designing Mobile Systems from Domain-Dependent Models

Peter Axel Nielsen and Jan Stage
Department of Computer Science, Aalborg University, Denmark
{pan, jans}@cs.aau.dk.

Abstract. Modelling of domain-dependent aspects is a key prerequisite for the design of software for mobile systems. Most mobile systems include a more or less advanced model of selected aspects of the domain in which they are used. This paper discusses the creation of such a model and its relevance for technical design of a mobile software system. Conventional approaches to modelling of context focus either on the application domain or the problem domain. These approaches are presented and their relevance for technical design of software for mobile systems is discussed. The paper also reports from an empirical study where a methodology that combines both of these approaches was introduced and employed for modelling of the domain-dependent aspects that were relevant for the design of a software component in a mobile telephone. The empirical study was conducted in two companies that produce software for mobile telephones. The resulting models of domain-dependent aspects are presented, and the experiences from the modelling process are discussed. It is concluded that a dual perspective based on both of the conventional approaches is relevant for capturing the aspects that are necessary for creating the domain-dependent models that are integrated in a mobile software system.

1 Introduction

Mobile software development challenge the modelling activities that precede the technical design of a software system. The context of a mobile system includes a broad spectrum of technical, physical, social and organizational aspects. Some of these aspects need to be built into the software. Selecting the aspects that are needed is becoming increasingly more complex with mobile systems than we have previously seen with more traditional information systems.

Lyytinen and Yoo [1] identify several drivers towards more and more mobile environments which they term the nomadic information environments. These include: mobility where the computing service follows the user rather than the user comes to the computing service, digital convergence where the standardization of services across platforms increases accessibility and networking, and mass scale where devices and mobile services will be available at significantly lower costs. In their survey of the new challenges for research on mobile systems, Lyytinen and Yoo [1] argue for several research agendas of which one is “How do we design and integrate sets of personalized mobile services that support users’ task execution in multiple social and physical contexts?” [1].

The context of a mobile system includes technical, physical, social and organizational domains. These dynamically changing domains in which mobile systems are used can be modelled for two reasons. First, we need to build selected aspects of these domains into the software. Without a representation of certain aspects from these domains, the mobile system cannot provide the services that users are requesting [2]. In this sense, a mobile software system is domain

dependent. For example, a GPS system needs an internal representation of the place where it is currently located. This representation is designed from models of selected aspects of the relevant domains in which the system is used. The GPS system is primarily dependent on the physical and technical domains. This kind of modelling is a primary concern for many conventional software engineering methods. Second, the process of designing a mobile system is based on an understanding of the domains in which it will be used. For the GPS system, the design of the physical appearance and the features of the system require an understanding of the activity of a prospective user and the social and organizational domains in which this user will be acting. Therefore, such aspects are modeled to ensure that the designers have them available when needed. This kind of modelling is a primary concern for many design methodologies such as participatory design, user centered design and contextual design.

Both of these approaches are necessary in sorftware development. Yet the challenges of each approach when developing software for mobile systems are staggering, and combining the two is even more demanding. Clemmensen and Nørbjerg [3] state that software engineering and the user-centered approaches that deal with human-computer interaction and usability are based on different theories and cannot be integrated. They argue that the standard software engineering methods are fairly ignorant of human-computer interaction issues, e.g. the Rational Unified Process [4] has little to say on design of human-computer interaction or usability design. It has, however, been argued that even though the two worlds are currently separated, it is realistic to try to bridge or fill that gap [5].

In this paper, we are specifically focusing on the former approach, i.e modeling of the domain-dependent aspects that will be built into the software. The purpose is to explore to what extent conventional approaches to modeling of domain-dependent aspects are relevant for and how they can be combined in the design of mobile software systems. In the following Section 2, we present conventional approaches to modeling of domain-dependent aspects. Section 3 presents the empirical study of domain-dependent modeling we have conducted in two software organizations. In Section 4, we present the results of the study with focus on the relevance of modeling approaches and the challenges of modeling domain-dependent aspects of mobile software systems. The broader aspects of the results are discussed in section 5 where we also compare to the results obtained in other studies related to mobile systems. Finally, Section 6 provides the conclusion.

2 Conventional Modelling Approaches

Software engineering methods that deal with modelling have different focus and perspective [6], and Avison and Fitzgerald [7] distinguish between process modelling, data modelling and object modelling.

2.1 Process Modelling

Process modelling base design of a software system on a model of the way in which a specific work process is carried out. The aim is that the software system will partly or fully automate the work process that is modeled.

A typical methodology within this approach is “Structured Analysis and Structured Design” (SASD). The first presentation of data flow modeling which was the core of this approach was DeMarco [8] that came out in 1979 and the first coherent method was Yourdon [9] from 1982. A more recent version is [10].

2.2 Data Modelling

Data modeling was developed as an abstract approach to database design as opposed to a direct focus on physical design. The classical reference is Date [11] that was fist published in 1977, and it is now in the eight edition. This approach focused on the data that were processed in the user organization, and it was closely related to the relational database as the implementation platform and diagramming techniques of which the entity-relationship model [12] was most prominent.

2.3 Object Modelling

The concept of classes and the idea of focusing on classes and objects stem from Simula programming language developed by Nygaard and Dahl, cf. [13]. In 1975, the concepts were also proposed for modelling through the Delta system description language that extended Simula [14]. In the early 1980s, Jackson launched the Jackson System Development method (JSD) for analysis and design [15]. The basic concept is "entity" but the term is clearly inspired by object-oriented thinking. Since classes and objects are not clearly differentiated, it is not possible to describe structural connections. The method's strength is that it introduces an event concept for describing entity dynamics. None of these methods gained substantial influence as modeling approaches.

The first significant methods for object-oriented analysis and design emerged around 1990. One stream of methods originated from object-oriented programming, e.g. [16] and [17]. A different stream of work originated from system development, e.g. [18], [19] and [20]. Some of these methods were joined into the "The Unified Software Process" [21] and later "The Rational Unified Process" (RUP) [4]. This method development was complemented with creation of a modeling language "Unified Modelling Language" (UML) of which the first version was standardized in 1998 [22].

For business applications as well as for technical applications object-oriented modelling has become a dominant paradigm. The advancement of UML and several strong object-oriented programming languages like Java, C++ and C# have pushed further in that direction.

2.4 Modelling for Technical Design

The three approaches approaches presented above all provide methodological guidance for turning the results of the modeling activities into a technical design. In this sense, they build the aspects that have been modeled into the system.

The process modeling approach takes its point of departure in the way users work. This relates more generally to a focus on this domain:

- *Application domain:* The individual persons or roles and the organization that administrates, monitors, or controls a problem domain.

The application domain is where the users are and do whatever they do when they use the system. For an air traffic control system, the application domain is in the control tower where the controllers perform their air traffic control. The controllers monitor the traffic on the screen, decide on interventions, and direct the flights in their air space.

With the process modeling approach, the domain-dependent aspects are elicited from the application domain and built into the system through the activities in which the software functions are designed.

The data modelling approach takes a different point of departure by focusing on the data that people work with in the user organization. It has been argued that this was a much more stable foundation for software design than the way in which the users worked, e.g. [15]. The data modeling approach relates more generally to a focus on this domain:

- *Problem domain:* The part of the context that is administrated, monitored or controlled by a system.

The problem domain is part of what is outside the system (i.e., in the context). For an air traffic control system the problem domain is that part of the context constituted by flights, departures, aircrafts, aircrafts' positions and trajectories, changed altitude, changed speed, etc. Everything that the controller in the tower needs to know about to control the air space effectively is in the problem domain. It is fundamental to air traffic control that the controller watches the aircrafts' positions and trajectories on a large monitor displaying data from the radar system rather than looking out the tower's windows with a pair of binoculars. The system creates and maintains the controllers' view of their air space and it is there crucial that the model of the air space (i.e., the problem domain) is in accordance with the controllers' professional language and competence. With the data modeling approach, the domain-dependent aspects are elicited from the problem domain and built into the system through the activities in which the database and the related software are designed.

The object modeling approach is more varied. Some of the methods, in particular the early ones, are focusing on the problem domain, e.g. [17], [18] and [19]. The RUP methods is completely opposite as it departs from use cases which are descriptions of the application domain.

Rumbaugh et al. [20] is the only classic object-oriented method that emphasizes both the problem domain and the application domain. Two of the three fundamental models are the class diagram, emphasizing the problem domain, and a description of functions by means of data-flow diagrams from structured analysis, emphasizing the application domain. This dual focus is an interesting and innovative approach. Unfortunately, the description of functions is not related to the object-oriented model. System developers with experience using the Rumbaugh method also point out that constructing the functional model is rarely worth while.

2.5 Software for Mobile Systems

The overview above illustrate that popular software engineering methods have a strong focus on technical aspects and the representation of information in the system. Yet they have very little in particular to offer in modelling context for mobile systems. Rational Unified Process [4], for example, offers several principles of which none address how to model the context of a mobile system. Microsoft Solutions Framework [23], as another example, offers a set of principles for software engineering, but, again, has nothing in particular to say on modeling the context of a mobile system.

The literature on human-computer interaction has a stronger emphasis on the context of computerized systems. The basic literature deals with user interface design from a general point of view, e.g. [24] and [25]. They provide extensive guidelines and techniques for user interaction design but

nothing specific on design of mobile systems and very little on modelling of domain-dependent aspect as a basis for technical design.

Some of the literature in human-computer interaction deals specifically with user interaction design for mobile systems. There is a general textbook on design of user interaction for mobile systems [14]. This textbook has a strong focus on mobile systems but significantly less on the modelling of domain-dependent aspects and very little emphasis on the relation to software engineering and technical design of software.

There is also a frowing body of literature on context awarteness. Some of this literature discusses definitions of 'context' and the implications for systems that are aware of their context. This group includes both conceptual work, e.g. [2], [27] and [28], practical guidelines for modelling and design, e.g. [29], [30] and [31], as well as examples of the process of analysing mobile activity [31] and designing context-aware mobile systems [32]. A common characteristic of this literature is that they work with various domain models, but there is very little about the relation to other models that are made for design of technical aspects, including the representation of information about the relevant domains.

There is some literature that deals with technical design of context-aware systems. For example, Anagnostopoulos et al. [33] formulate a set of requirements for what should be modelled in designing mobile systems: context data acquisition (e.g., from sensors), context data aggregation, context data consistency, context data discovery, context data query, context data adaptation, context data reasoning, context data quality indicators, and context data integration.

Henricksen et al. [34] start by recognizing that what is unique to modelling context for mobile systems: information on context is temporal, it is imperfect, context has several representations, and context information is highly interrelated. Thedy then go on to suggest a graphical modelling language that allow them to use three entity types 'people', 'device', and 'communication channel'. In addition, they can model association between entities like 'static' or 'temporal', dependencies between associations, and finally qualities of dependencies like 'accuracy' and 'certainty'.

Lei and Zhang [35] apply the theory of conceptual graphs to mobile context modelling. All items in the context are monitored and the conceptual graphs model this as simple graphs associated with rules and constraints. Baumeister et al. [36] extend UML with mobile objects, locations, and mobile activity. They formulate the extension in terms of stereotypes and end by providing a modified metamodel for UML.

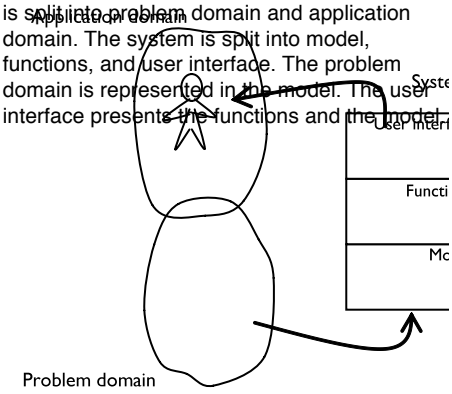

Fig. 1. The system and its context. The context is split into problem domain and application domain. The system is split into model, functions, and user interface. The problem domain is represented in the model. The user interface presents the functions and the model.

3 Empirical Study

We have conducted an empirical study of the way in which conventional approaches to modeling of domain-dependent aspects can be combined in the design of mobile software systems. In this section, we describe the method of the study.

3.1 Setting

The study was carried out as part of a training course that was conducted for two companies that develop software for mobile telephones. In both companies, it was part of a method deployment effort. The purpose of the course was to teach them how to model domain-dependent aspects and how the resulting models could be integrated in software design. In one of the companies, the course was conducted once, while in the other it was conducted twice for two different departments.

In both cases, the course was held in the company's own buildings.

3.2 Participants

The participants in the courses were software developers from the two companies. The first company was a mobile telephone producer, where the participants came from a software development group. The second company was a producer of specific components for a mobile telephone. In both cases, there were 20-25 participants each time we held the course. The courses were held by the authors of this article. The courses were planned in collaboration and held by one of the authors. Both authors had considerable experience in giving analysis and design courses for software developers.

3.3 Methodology

The course was based on an object-oriented method known as Object-Oriented Analysis & Design (OOA&D), described in [37]. The basic modeling approach of the method is also presented in [38]. The method combines modeling of the problem and application domains.

The combination of the problem and application domain models is based on a simple architectural model of a general computerized system, cf. figure 1. The system contains a representation of the problem domain; this representation is also known as the model. The model keeps track of objects and events in the problem domain. This can be illustrated with the

air traffica control system: When an air craft enters the air space, the model gets a new object that represents the aircraft. When the radar detects that an aircraft has changed altitude and position, the object that represents this aircraft in the model will also change altitude and position. The users in the application domain get access to the model through the functions. When a controller wants to see the data from the radar on her monitor she utilizes a function that reads the relevant objects in the model. There is a whole range of functions that the controller may utilize at her discretion and by these functions update or read the model's objects. A warning that two aircrafts are too close is also issued by a function assisting the controller.

The user interface provides access to functions and handle input and output of information about the problem domain. The model objects are presented in a way that allows the users to view the state of affairs in the problem domain through the system. The functions are presented in a way that allows the users to view and activate the possible operations the system offers to manipulate the objects in the model.

With OOA&D [28], the problem domain is modelled in terms of objects and events. The model is expressed in UML as a class diagram with all problem domain classes and a statechart diagram describing the dynamics of each problem domain class. These domain-dependent aspects are later designed into the computer system model. The model is a representation of the problem domain and it tracks objects and events in the problem domain. The system only registers events if information about them is required by the users.

The application domain is modeled by use cases and the functions of the system. The functions are modelled through functional specification.

3.4 Research Method

On the overall level, our research approach was a combination of action research and case study research. This combination is commonly referred to as action case [39]. The study was action research in the sense that we were part of a situation in a software company where we helped diagnose the current development practice, devised an action plan, implemented the action plan, and reflected on the lessons learned from this [40]. Action research employs a dual learning cycle [41] in which the practitioners we collaborated with were occupied with solving a number of specific problems with how to model domain-dependent aspects as well as the interior of a mobile application and we, the action researchers, focused our attention on the modelling method. Action research must address, utilise and make a contribution relative to a body of knowledge; in particular it must declare the framework in which learning is to take place [42], [43]. The body of knowledge that we have contributed to is that of methods for developing and modelling mobile applications in the specific software companies.

The study was case study research in the sense that we have not been involved in the software development project itself. We have worked with the software developers and we have through our interactions gained insight into the development context and the larger activities and company setting in which the development took place. Towards these issues we have been observers rather than actors [44].

3.5Procedure

The course was conducted in 3 full days. The first two days focused on modeling, with half a day as introduction to the method and the basic concepts, a full day on problem domain modeling and half a day on application domain modeling. The third day focused on system design where the main issue was the transformation of the problem and application domain models into an overall design of the software.

The course comprised a mix of lectures and exercises. The exercise was to model and design an FM radio as an integrated component in a mobile telephone. This is an application that is available in some mobile telephones. It enables the user of the mobile telephone to listen to radio while the telephone functionality is not in use. The heart of the system is an FM radio receiver. The user can interact with certain information and functions relating to the FM radio through the telephone's user interface.

After each lecture, there was an exercise in the same topic. The participants worked with the exercise in small groups of 3 to 5 participants. The exercise was completed with a short presentation of the results to the other groups.

When the course was completed, the authors made models of the problem and application domains. These models were based on the models made by the participants and the knowledge acquired durting the course. It was presented as a best solution to the exercise, and we collated comments from the participants. This model was refined from course to course.

After the courses, the authors collaborated with the companies to integrate the method in the developers' work. That part of the work is not described in this paper.

3.6Data Collection

During the course, the lecturer made notes about the questions that were asked and the difficulties that the developers faced in learning the method and using it for modeling domain-dependent aspects. The notes also included questions that were asked by the participants in the lectures and in particular when the Notes and exercises

We also collected all the models that were made during the courses.

Finally, we have documented all comments that came up in the evaluation of each course.

We have analysed the empirical data by focussing on issues related to the notion of context. We have traced through the models and analysed the various modelling concepts. We have also analyzed and categorized the questions that were noted during the course and the issues that were emphasized in the final evaluation of each course. The purpose of the analysis was to identify challenges when modelling the domain-dependent aspects of a mobile system and the solutions we employed.

4 Results

This section presents the results of the empirical study. First, we present domain models that were produced in the study. Second, we provide an overview of the lessons learned.

TABLE 1. SYSTEM DEFINITION FOR THE FM RADIO.

F	**Functionality**	Play radio, change radio settings, automatic tuning (search), manual tuning, define station presets
A	**Application domain**	Listen to radio and change station without interfering with normal use

		of the telephone
C	**Conditions**	The system should use the API, it must cooperate with the media server, and it must turn off automatically when the GSM part becomes active
T	**Technology**	Specifications of implementation platform, execution platform, interfaces to TEA, GSM part, and media server
O	**Objects**	Radio state, Station(s)
R	**Responsibility**	Automatic cooperation with the GSM part, manual use of the radio

4.1 Domain Models

The first result of using OOA&D is a system definition that provides an overall delineation of the problem and application domains as a whole. The system definition for the FM radio is summarized in Table 1.

TABLE 2. CLASSES AND EVENTS IN THE PROBLEM DOMAIN.

	Radio	**GSM part**	**Radio Chip**	**Station**	**Preset Station**	**Current Station**
GSM part activated	✓	✓	✓			
GSM part deactivated	✓	✓	✓			
Radio turned on	✓		✓			
Radio turned off	✓		✓			
Radio setting changed	✓					
Radio chip turned on	✓		✓			
Radio chip turned off	✓		✓			
Search activated			✓			
Search completed	✓		✓	✓		
Frequency entered	✓		✓			✓
Preset station defined					✓	
Preset station selected	✓		✓		✓	✓

Problem Domain Model

Within the frame of the system definition, the classes and events in the problem domain were selected. They are shown in Table 2, where all classes are listed in the horizontal dimension and all events in the vertical dimension. A checkmark in the table means that the class is involved in the event.

The structural relations between the problem domain classes are shown in the class diagram shown in
Fig. 2, in the UML notation.

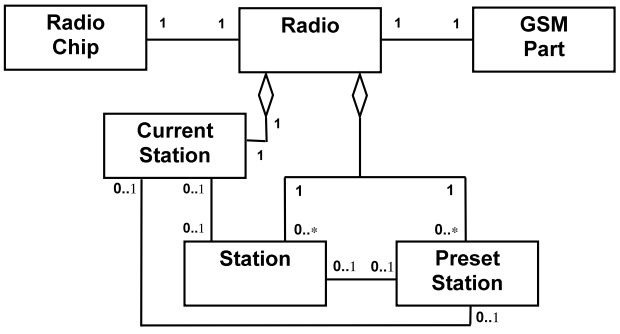

Fig. 2. Class diagram for the problem domain.

For each of the classes in the class diagram, there was a statechart diagram describing the dynamic behaviour of objects from this class.

TABLE 3. EXAMPLE OF A USE CASE FOR THE FM RADIO.

Use case	The radion is turned on by the user when he wants to listen to radio on the mobile telephone. The user then selects a preset station. If there is no signal on the frequency that is preset, an error message appears. Otherwise, the radio will play the station on the channel. If the telephone receives a call, the radio is muted. When the call is completed, the radio resumes.
Objects	Radio, Preset Station, Current Station
Functions	Turn radion on, Select preset station, Display error, Mute radio, Resume radio

Application Domain Model

The key elements of the application domain were also described. The first part of this is descriptions of all actors and a set of use cases. For the FM radio, there were two actors: the user and the GSM part of the telephone. The user is anobvious actor. The GSM part was included as an actor, because it can activate functions, e.g. mute the radio when a call is received.

The use cases describe how a prospective user will apply the system for listening to radio. Table 3 shows an example of a use case.

TABLE 4. EXCERPT FROM FUNCTION LIST FOR THE FM RADIO.

Function	Complexity	Type
Turn radio on	Simple	Update
Turn radio off	Simple	Update
Select preset station	Simple	Read + Update
Mute radio	Medium	Signal
Sech for stations	Complex	Update

4.2 Experience from the Modelling Process

This section presents the second part of results of our action case study. Here, the emphasis is on the experience that was gained when using the method to create the domain-depdendent models that were presented above.

The System Definition

The purpose of the system definition is to delineate the context of the system considered on an overall level. The following challenges were experienced with the system definition:

- Vehicle for keeping focus
- Defining granularity
- Point of view
- Application domain versus functions

It was difficult to make a good system definition for the FM radio. The reason is that it expresses key decisions that are difficult to make. However, once it was made, it turned out to be a very powerful tool for

keeping focus during the whole modelling activity. Sometimes, the participants forgot to use the system definition when they discussed an issue related to the limits of the problem or application domains.
The system definition inherently involves defining the granularity in the model of the system's context. For example, the system definition mentions key objects, and this influences the considerations made during problem domain analysis. The participants had difficulties selecting a relevant level of granularity. They could either model a single component of the system or the whole system. When they focussed on the whole system, they also had a choice of seeing it as one or a few complex components or a larger number of simpler components. The solution employed was to see the system as one unit with relations to other systems.
A system can always be described from two opposite points of view. Either from the outside where it is considered as a black box; or from the inside where the focus is on its components. For example, the participants could describe the relation between the user and the mobile telephone with the radio, or they could describe the radio as a component and its interplay with other components. The participants shifted between these points of view. For the system definition, it was suggested to see the system as a black box and, therefore, not deal with its components.
The system definition has the application domain as one of its six elements. The aim of this is to emphasize who the users are and what their role is in the user organization. The participants had problems describing this element. In most cases there was no clear distinction between application domain and functions. This is also reflected in the system definition shown in Table 1. A mobile system is usually operated by a single user. Therefore, the application domain becomes very simple, and the developers had difficulties believing that it could be that simple.
This challenge emphasizes a key difference between conventional and mobile systems. For a stationary application, it is often useful to describe the domain where it is applied. For a mobile system, that is not possible because of the mobile nature of the system. This follows from the definition of a mobile system that was provided in the introduction. For a mobile system, it is more interesting to describe the application situations, i.e. the situations in which the device is used.

The Problem Domain Model

The problem domain model describes the part of the context that the users administer, monitor and control by means of the system. The modelling of this faced the following challenges:

- Identifying classes
- Identifying events
- No physical counterparts
- Classes with a single object
- Classes versus functions
- Users and clients
- Structural relations

Identification of the key classes was relatively straightforward. It was difficult to get started, but once the first candidates were identified, the participants suggested several more. The criterion used was whether the

system would need to maintain information about objects from a class that was suggested. In addition, a class must be manageable. Some classes may be so small that they are really only representing an attribute. An example is Current Station. It is still included in the model because it represents a key concept that should appear in the model. We identified classes of a category that is not seen in conventional systems. The FM radio depends on two other components that are modelled as classes: GSM Part and Radio Chip in Fig. 2. In order to interact with these components, the FM radio needs to maintain updated information about their current state.

It turned out to be much more difficult to identify events. We ended up modelling events that represent user actions and state changes for related components that the system needs to register or respond to. The identification of events involved significant discussion of the way in which the system can detect that these events have happened.

The problem with events and to some extent also classes originates from the nature of the context. With a conventional system, there is often an existing physical context with tangible objects that we can model. A classical example is an inventory system, where the context includes the warehouse and the physical objects in it. Another example is air traffic control, where the context includes physical objects such as airplanes. For the typical mobile system, there is usually not an existing context with physical objects. The key classes in the class diagram for the FM radio model virtual objects that only exist in the computer system. We solved this by considering to what extent the system should administer, monitor or control other components that were part of the device. These became classes.

In object-oriented methods, a class can be defined as a description that is common for a set of objects. Moreover, it is often suggested that classes with only a single object are very unusual. In our case, four out of six classes had only one object (Radio, GSM Part, Radio Chip and Current Station). We have seen the same with other applications for mobile devices. Thus single object classes are more common. Other examples of potentially relevant single object classes are screen, loudspeaker and headset but also more abstract items like a session and a connection.

The participants modeled the problem domain before they modeled the application domain. A consequence of this order was that they defined problem domain classes that really reflected requirements to functionality and thereby pertained to the application domain model. We solved this by emphasizing the criterion: “Do you want to maintain information about this?” This problem is solved with experience, because more experienced designers model the problem and application domains in parallel.

The context of the mobile device included clients, which is a generic term we introduced to cover both users and other components that will use the system we are modeling. The question was whether these clients should be included in the model of the problem domain. The general answer is that they should be in the model if we need to register information about them. For the FM radio it was considered as a possibility that different users could have their individual sets of preset stations. In that case, User would have been a class in the model of the problem domain.

The model of the problem domain can employ the following object-oriented structures: generalization, aggregation and association. In our study, it was quite easy to explain these means of expression in general, but it was hard to relate them to their "actual counterparts" when the system was running. It was especially difficult with aggregation and association. This was solved by focusing on the 'has-a' definition of aggregation. In addition, we used construction and destruction of objects to differentiate between associated and aggregated objects.

The Application Domain Model

The purpose of the application domain model is to describe how the model of the problem domain is applied by users and other systems. Key elements in the model are use cases and a complete list of system functions. The modeling of this faced the following challenges:

- Identifying users
- Generating use cases

The conventional approach is to define the users of the system as the people who are working in the application domain. Above, it was emphasized that it was impossible to identify a specific application domain for the FM radio. As a consequence, we had no simple way of identifying the users of the system. In addition, many components in a mobile system are applied by other components. Some of these components may then have users, and a difficult question was whether we should describe the component as the user or the user of that external component as the users of the system in question. To solve this, we introduced the notion of client which includes users as well as other systems or components. Thereby, we emphasized the direct user of the system considered.

A related problem was that it was difficult to generate use cases. They are usually defined by identifying tasks in the application domain and then selecting those that will be supported by the system. This is basically a top-down approach. The difficulty of describing the application domain for the FM radio implied that we could not use that approach. Instead, we tried a bottom-up approach, where the participants described a set of activities that would involve use of the system.

For the system definition, it was suggested to focus on application situations instead of the application domain. This might also make it easier to generate relevant use cases. However, we did not try this idea out in our study.

Modelling System Behaviour

The problem domain model is by nature static. During the modelling of the context for the FM radio, certain dynamic issues were considered. These issues were:

- Detecting event occurrences
- Handling errors
- Connected devices

Events are described as part of the problem domain model. In design, it is decided how they are represented in the model and presented on the user

interface. When the system is running, the model should be updated each time an event occurs. Therefore, we need to consider for each event how we can detect an occurrence. This issue is important already in the modelling activity. If an event occurrence cannot be detected, the event should not be in our model.

In the modelling of the context of the FM radio, errors turned out to be a very important issue. These errors were often complex and related both to the problem domain, the application domain and the functions of the system. With a classical administrative system, errors are handled by humans. Therefore, it is not as important to include them in the model. This was different for the mobile system that we modelled. Here, the system would need to handle errors, and therefore they should be described. Based on this experience, we would suggest that potential conflicts between systems are identified on an overall level in the system definition as part of the description of the technical platform.

The system needs information about the devices that are connected. If the set of connected devices can change dynamically, they need to be registered by the system. In our case, we turned out to have a static set of connected devices. Therefore, we did not inquire into this aspect.

5 Discussion

The lessons we have learned from the action case study are twofold. First, the participants have managed to model the context of their mobile system by means of the concepts we offered them from OOA&D. They modelled the domain-dependent aspects they will need to represent in the system and explicitly leaving out quite many aspects of the context that the mobile system need not be concerned with. The distinction between problem domain and application domain helped them decide for their system what is part of the dynamically changing context which need to be modelled. The concepts of object and event helped them figuring out how to represent that part of the context in the model. The model contains objects, but more importantly the objects experience events that represent the defined changes.

Second, the participants did meet several challenges as explained in section 4.2. It is interesting to note that the challenges were concerned with how to apply the distinctions and concepts. When the participants had understood how to apply the distinction and the concepts, they had no problems with modelling by means of a subset of the modelling language UML 2.0. We therefore contend that the challenges for modeling domain-dependent aspects of mobile systems is not really a matter of extending the existing modelling language irrespective of whether that is based on conceptual graph theory or on UML.

The lessons learned from the action case study emphasize two other issues. First, the approaches to modelling context for mobile systems described in section 2 and 3 have something to offer for both the concern for representation and the concern for presentation. The results from the action in section 4 show in detail that modelling is all about deciding what should be represented in the model. That should be clear from the described experience with how to identify objects and how to identify events in subsection 4.2. The linkage to concern for presentation is as follows. The very reason to include a particular object or a particular event is that it needs to be presented in the user interface. The class ‘Current Station’ is in the model as a representation of the problem domain. It is there because we want to be able to show in the display what the current station is. Similarly, the event

‘Frequency entered’ is there because we want to be able to show on the display that a frequency has been entered for the current station. The linkage between the representation and the presentation is in one sense intrinsic because we cannot show any item or behaviour in the problem domain that is not represented in the model. The linkage is also in some sense extrinsic because there is separation of concerns in the system’s architecture. The architecture is layered and how the problem domain is represented is hidden for the above layers. Vice versa, how the problem domain items and behaviour is presented is hidden for the model.
Second, the modelling of domain-dependent aspects of mobile systems as we have explained it in section 2 and 3 is an improvement compared to other software engineering methods like RUP [4] and MSF [23] because: (a) it recognizes a context that is outside the system that needs to be represented within the system, and (b) it offers a better linkage to the design of the human-computer interaction. It is also an improvement compared to the human-computer interaction methods because: (a) it points directly at what in the context that needs to be presented, (b) it offers a better linkage to software engineering, and (c) it utilizes a clear, layered software architecture that separates concerns between the model (what is represented) and the user interface (how it is presented).
For a mobile system adaptability is a main issue. When Anagnostopoulos et al. [33] state the modelling requirement that “the application should be capable of adapting its behavior according to contextual information” little has really been said as any non-trivial software system should have reasonable responses to its stimuli which in the case of a mobile system will encompass contextual changes. At a more specific level adaptability is easier said than implemented. The approach to modeling that we have presented here is particularly directed at deciding exactly which contextual changes that are taken to be important to the system. This means that the model will be adaptive towards the changes that are already modeled as events in problem domain. It also means that the system is incapable of adapting to whatever changes that may occur in the context which we have not modeled. At one level there is little surprise in this as the system can only perform according to an algorithmic specification. At another level it is often forgotten in the process of modeling that a system can only adapt to events that have already been foreseen.
Adaptability towards changes in the problem domain is thus dealt with by modeling the changes as events. Adaptability towards changes in the application domain and in the underlying technical resources has to be dealt with in a slightly different way. Changes in the application domain have to be handled by providing several sets of interaction. That means we will have to model this in more than one set of use cases. A particular set of use cases will then be designed for a particular purpose (or situation as discussed above). The user’s purpose (or situation) changes the available interaction should change similarly. To achieve that, the user’s purpose (or situation) has to be modeled as part of the problem domain, i.e., we need to know about the user’s task, purpose or situation.
Changes in the underlying technical resources have to be handled by encapsulating the different underlying resources. An example of that is that in one situation we are connected to the world by a Bluetooth connection to a stationary device and in another situation we are connected to the world by a WiFi connection. Again, we need to know about which technical resources are with reach and they should thus be modeled as part of the problem domain. It then

becomes an event when we are entering Bluetooth reach and another event when we are leaving.

6 Conclusion

This paper addresses how we can model domain-dependent aspects of mobile systems with the purpose of providing a basis for designing the system. We have surveyed existing research literature on this question. The existing literature focuses either on the problem domain or the application domain. For designing a mobile system, this is not adequate.

We have presented results from an empirical study of domain modeling in relation to a specific mobile software system. We have modeled the application and problem domains for this system with the aim of providing a useful basis for technical design of the mobile system. The application has been used in an action case study where we have enquired into teaching and deployment of a modelling method in a large group of software developers in mobile software companies. From the action case study we have presented both the results in terms of the models we arrived at and in terms of the process we went through. The resulting models show that the problem domain for the example case has been modelled in terms of classes and events. The process experience emphasizes the challenges faced by software developers whehn modeling domain-dependent aspects of a mobile system.

The method we have applied in the action case study and reported on in this paper suggests modelling of domain-depdendent apsepcts of mobile systems based on ther following five principles:

- The modelling of context is based on a fundamental distinction between the problem domain and the application domain.
- The starting point is a system definition that assists in defining the system's scope and delimits the problem and application domains.
- The problem domain is modelled in terms of the basic concepts 'object' and 'events' and described in a class diagram and a set of statechart diagrams.
- The application domain is modelled in terms of 'use cases' and 'functions' and described in related specifications.
- The model and its selection of objects and events are useful for both design of representation and presentation in a software system.

ACKNOWLEDGMENTS

The research we report in this paper extends ideas that we have developed in close collaboration with Lars Mathiassen and Andreas Munk-Madsen through the decade 1990-2000. For the empirical research that we build on we are greatly in depth to Siemens Mobile, RTX Telecom and the software developers we have worked with.

REFERENCES

[1] Lyytinen, K. and Y. Yoo, Research Commentary: The Next Wave of Nomadic Computing. Information Systems Research, 2002. 13(4): p. 377-388.

[2] Dey, A.K. and G.D. Abowd, Towards a Better Understanding of Context and Context-Awareness. 1999, Georgia Institute of Technology.

[3] Clemmensen, T. and J. Nørbjerg, Separation in theory, coordination in practice - teaching HCI and SE. Software Process: Improvement and Practice, 2003. 8(2): p. 99-110.

[4] Kruchten, P., The Rational Unified Process: An Introduction. 3 ed. Object Technology Series, ed. G. Booch, I. Jacobsson, and J. Rumbaugh. 2003: Addison-Wesley Pearson Education.

[5] Seffah, A. and E. Metzker, The obstacles and myths of usability and software engineering. Communications of the ACM, 2004. 47(12): p. 71-76.

[6] Nielsen, P.A., Reflections on development methods for information systems: a set of distinctions between methods. Office, Technology and People, 1989. 5(2): p. 81-104.

[7] Avison, D. and G. Fitzgerald, Information Systems Development: Methodologies, Techniques and Tools. 3 ed. 2002, London: McGraw-Hill.

[8] DeMarco, T. (1979). Structured Analysis and System Specification. Yourdon Inc. & Prentice-Hall, Englewood Cliffs, New Jersey.

[9] Yourdon, E. (1982). Managing the System Life Cycle. Yourdon Inc., New York.

[10] Yourdon, E. (1989). Modern Structured Analysis. Prentice-Hall, New York.

[11] Date, C. J. 1977 An Introduction to Database Systems (The Systems Programming Series). Addison

Wesley Longman Publishing Co., Inc.
[12] Peter Chen (March 1976). "The Entity-Relationship Model - Toward a Unified View of Data". *ACM Transactions on Database Systems* 1 (1): 9-36.
[13] Dahl, O.-J., B. Myhrhaug & K. Nygaard, 1971. SIMULA 67 Common Base Language. Publikasjon nr. S-22, Norsk Regnesentral, Oslo.
[14] Holbæk-Hanssen, E., P. Håndlykken & K. Nygaard, 1975. System Description and the DELTA Language, Norsk Regnesentral.
[15] Jackson, M., 1983. System Development. Prentice-Hall, Englewood Cliffs, New Jersey.
[16] Meyer, B., 1988. Object-Oriented Software Construction. Prentice Hall, Hemel Hempstead.
[17] Booch, G., 1994. Object-Oriented Analysis and Design with Applications, second edition. Benjamin/Cummings, Redwood City, California.
[18] Coad, P. & E. Yourdon, 1991a. Object-Oriented Analysis, second edition. Prentice-Hall, Englewood Cliffs, New Jersey.
[19] Coad, P. & E. Yourdon, 1991b. Object-Oriented Design. Prentice-Hall, Englewood Cliffs, New Jersey.
[20] Rumbaugh, J., M. Blaha, W. Premerlani, S. Eddy & W. Lorensen, 1991. Object-Oriented Modelling and Design. Prentice-Hall, Englewood Cliffs, New Jersey.
[21] Jacobson, I., G. Booch & J. Rumbaugh, 1999. The Unified Software Development Process. Addison-Wesley, Reading, Massachusetts.
[22] Object Management Group, 1998. Unified Modeling Language Specification. Framingham, Massachusetts.
[23] Turner, M.S.V., Microsoft® Solutions Framework Essentials 2006: Microsoft Corporation.
[24] Dix, A., et al., Human-Computer Interaction. 3 ed. 1998, London: Prentice-Hall.
[25] Preece, J., Y. Rogers, and H. Sharp, Interaction Design: Beyond Human-Computer Interaction. 2002, New York: John Wiley and Sons.
[26] Jones, M. and G. Marsden, Mobile Interaction Design. 2006: Wiley.
[27] Agre, P., Changing Places: Contexts of Awareness in Computing. Human-Computer Interaction, 2001. 16(2): p. 177-192.
[28] Dourish, P., Seeking a foundation for context-aware computing. Human-Computer Interaction, 2001. 16(2): p. 229-241.
[29] Dey, A.K., G.D. Abowd, and D. Salber, A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications. Human-Computer Interaction, 2001. 16(2): p. 97-166.
[30] Graham, C. and J. Kjeldskov. Indexical Representations for Context-Aware Mobile Devices. in IADIS e-Society 2003. 2003. Lisbon, Portugal: IADIS.
[31] Morse, D., S. Armstrong, and A.K. Dey. The What, Who, Where, When, Why and How of Context-Awareness. in CHI 2000. 2000: ACM.
[32] Kjeldskov, J. and J. Paay. Augmenting the City: The Design of a Context-Aware Mobile Web Site. in DUX 2005. 2005. San Francisco, CA, USA: ACM.
[33] Anagnostopoulos, C.B., A. Tsounis, and S. Hadjiefthymiades, Context Awareness in Mobile Computing Environments. Wireless Personal Communications Journal, 2006.
[34] Henricksen, K., J. Indulska, and A. Rakotonirainy. Modeling Context Information in Pervasive Computing Systems. in Pervasive 2002. 2002: Springer-Verlag.
[35] Lei, S. and K. Zhang. Mobile Context Modelling using Conceptual Graphs. in Wireless And Mobile Computing, Networking and Communications, 2005. (WiMob'2005) 2005: IEEE.
[36] Baumeister, H., et al. Extending Activity Diagrams to Model Mobile Systems. in NODe 2002. 2003: Springer-Verlag.
[37] Mathiassen, L., et al., Object-Oriented Analysis & Design. 2000, Aalborg: Marko Publishers.
[38] Mathiassen, L., et al. Modelling Events in Object-Oriented Analysis. in International Conference on Object Oriented Information Systems. 1994. London, U.K.: Springer.
[39] Braa, K. and R. Vidgen, Interpretation, Intervention and Reduction in the Organizatioanl Laboratory: A Framework for In-context Information Systems Research. Accounting, Management and Information Technologies, 1999. 9: p. 25-47.
[40] Avison, D., et al., Action Research. Communications of the ACM, 1999. 42(1): p. 94-97.
[41] McKay, J. and P. Marshall, The dual imperatives of action research. Information Technology & People, 2001. 14(1): p. 46-59.
[42] Iversen, J.H., L. Mathiassen, and P.A. Nielsen, Managing Risk in Software Process Improvement: An Action Research Approach. MIS Quarterly, 2004. 28(3): p. 395-433.
[43] Nielsen, P.A., IS Action Research and Its Criteria, in Information Systems Action Research: Bridging the Industry-University Technology Gap, N.F. Kock, Jr., Editor. 2006, Springer Verlag: New York.
[44] Yin, R., Case study research: Design and methods. 2 ed. 1994, Beverly Hills, CA: Sage Publishing.