Diploma Thesis

Context Modeling in the Domain of Ambient Intelligent Production Environments

A context engine to support adaptive user interfaces using the example of the SmartMote

Kai Bizik

October 26, 2010

Supervisors:
Prof. Dr. Dr. h. c. H. Dieter Rombach
Dipl.-Inf. Kai Breiner
Dipl.-Inf. Marc Seißler*

AG Software Engineering: Processes and Measurement
Prof. Dr. Dr. h. c. H. Dieter Rombach
Technische Universität Kaiserslautern
Fachbereich Informatik

*Lehrstuhl für Produktionautomatisierung

Erklärung

Hiermit erkläre ich, dass ich die vorliegende Diplomarbeit mit dem Thema "Context Modeling in the Domain of Ambient Intelligent Production Environments" selbstständig und nur unter Zuhilfenahme der angegebenen Literatur angefertigt habe. Inhaltliche oder wörtliche Zitate sind als solche gekennzeichnet und im Quellenverzeichnis aufgeführt.

Kaiserslautern, October 26, 2010

Kai Bizik

Abstract

Context is that which surrounds, and gives meaning to, something else.

- The Free On-line Dictionary of Computing -

Ever since Mark Weiser's vision of *Ubiquitous Computing* the importance of context has increased in the computer science domain. Future *Ambient Intelligent Environments* will assist humans in their everyday activities, even without them being constantly aware of it. Objects in such environments will have small computers embedded into them which have the ability to predict human needs from the current context and adapt their behavior accordingly. This vision equally applies to future production environments. In modern factories workers and technical staff members are confronted with a multitude of devices from various manufacturers, all with different user interfaces, interaction concepts and degrees of complexity. Production processes are highly dynamic, whole modules can be exchanged or restructured. Both factors force users to continuously change their mental model of the environment. This complicates their workflows and leads to avoidable user errors or slips in judgement. In an *Ambient Intelligent Production Environment* these challenges have to be approached.

The SmartMote is a universal control device for ambient intelligent production environments like the *SmartFactory*$^{\mathbf{KL}}$. It copes with the problems mentioned above by integrating all the user interfaces into a single, holistic and mobile device. Following an automated *Model-Based User Interface Development* (MBUID) process it generates a fully functional graphical user interface from an abstract task-based description of the environment during run-time.

This work introduces an approach to integrating context, namely the user's location, as an adaptation basis into the MBUID process. A *Context Model* is specified, which stores location information in a *formal* and *precise* way. Connected sensors continuously update the model with new values. The model is complemented by a *reasoning* component which uses an extensible set of rules. These rules are used to derive more abstract context information from basic sensor data and for providing this information to the MBUID process. The feasibility of the approach is shown by using the example of *Interaction Zones*, which let developers describe different task models depending on the user's location. Using the context model to determine when a user enters or leaves a zone, the generator can adapt the graphical user interface accordingly.

Context-awareness and the potential to adapt to the current context of use are key requirements of applications in ambient intelligent environments. The approach presented here provides a clear procedure and extension scheme for the consideration of additional context types. As context has significant influence on the overall *User Experience*, this results not only in a better usefulness, but also in an improved usability of the SmartMote.

Danksagung

An dieser Stelle möchte ich mich zunächst für die vielen Eindrücke bedanken, die ich im Rahmen meiner Tätigkeit als wissenschaftliche Hilfskraft in den letzten Jahren in der AG Software Engineering sammeln durfte. Die Mitarbeit an verschiedenen Projekten und die Einführung in wissenschaftliche Arbeitsweisen waren und sind ein großer Erfahrungsgewinn.

Für die fachliche Betreuung der Diplomarbeit bedanke ich mich bei Herrn Dipl.-Inf. Kai Breiner und Herrn Dipl.-Inf. Marc Seißler, die stets für Fragen zur Verfügung standen und mich nicht nur im Rahmen dieser Arbeit betreut haben, sondern auch darüber hinaus Einblicke in die Strukturen und Zusammenarbeit der Arbeitsgruppen ermöglicht haben.

Besonderer Dank gebührt auch meinen Eltern und Geschwistern, die mich während meines Studiums in jederlei Hinsicht unterstützt haben. Ich genieße es sehr, einen solchen familiären Rückhalt zu haben.

Weiterhin bedanke ich mich bei Herrn Friedemann Werner für das Korrekturlesen großer Teile der Arbeit, auch noch kurz vor der Abgabe.

Danke auch allen guten Freunden, die ich während meines Studiums gefunden habe und die mir während der Höhen und Tiefen dieser Zeit zur Seite standen.

Der abschließende Dank geht an die Mitglieder der FeG Nord, des Marburger Kreises und allen, die mir in der Diplomarbeitsphase mit Verständnis, Zuspruch und Gebet geholfen haben. Ich bin froh einen Gott zu haben, der mir solche Leute zur Seite stellt und auf den ich mein Vertrauen setzen kann.

Kaiserslautern, October 26, 2010

Kai Bizik

Contents

1. Introduction

During the last years the mainstream understanding of a computer has changed dramatically. While the computer was usually pictured as a fixed apparatus with a mouse, keyboard and CRT monitor during the 90s, users nowadays rely on a multitude of small, mobile devices. It is not uncommon that one person uses a laptop, smartphone or personal digital assistant in addition to his home computer (cf. Figure 1.1). Consumer products like MP3 players and eBook readers have also become widely available and accepted. The variety and mobility of today's devices allow users to freely move their computational tasks from one environment to another. Most mobile devices are able to connect to the internet via wireless communication. The notion of "information at your fingertips" [Wei91] seems to be more up to date than ever before. The advancing miniaturization will enable the emergence of *Ambient Intelligent Environments* in which smart objects capture human needs and provide intelligent support for their everyday activities. To be able to do this they need to know about their environment. Data collected from sensors is used to adapt to the current *Context of Use* leading to an overall improvement in the Human-Computer-Interaction (HCI).

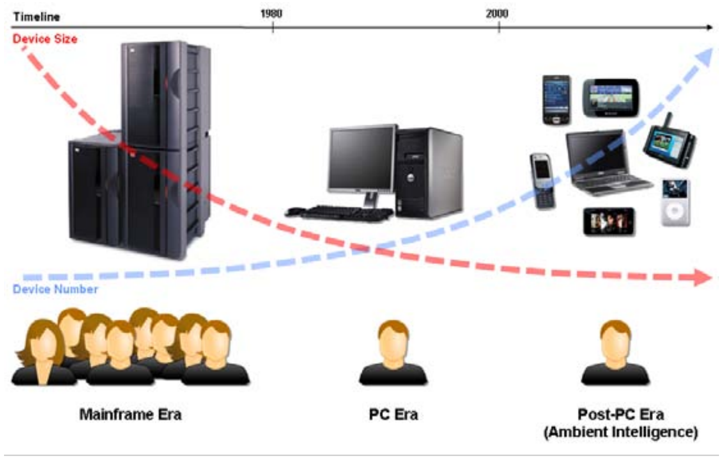


Figure 1.1.: Trends in computing [Tra09, Wei20]

This is also true in the domain of industrial production. The multitude of devices in modern factories leads to complex workflows. In such highly distributed environments users have to interact with many different devices from different manufacturers, all providing different user interfaces that follow different interaction concepts and offer different look&feel. This confrontation increases the likelihood of avoidable user errors [BMGM09]. The high degree of flexibility poses a second challenge. Entire production modules can be removed, replaced or reorganized [Zö9]. This can happen manually or automatically as a reaction to malfunctions.

A self-organizing factory is able to structurally reorganize its modules to ensure the production process. User interfaces and control concepts have to reflect these changes in an intuitive form.

The principle of *Universal Control* in combination with a *Model-based User Interface Development* (MBUID) [Pue97] process can be used to meet these challenges. Combining the user interfaces of available devices into a single and mobile *Universal Control Device* (UCD) helps to minimize user errors and simplifies users' workflows [BGSG10].

This work introduces a framework that provides context information for an MBUID process. Using the example of the SmartMote, a universal control device for the *SmartFactory*$^{\mathbf{KL}}$, it is demonstrated how the current context of use can be utilized for the run-time adaptation of an automatically generated user interface. This results in the ability to accurately represent the current state of the environment which is summarized in the attribute of *Real World Matching*, one of the main influence factors on the overall *User Experience* [Arn06].

The feasibility of the developed framework is demonstrated by implementing location-aware behavior in the SmartMote. The abstract task model used in the SmartMote's user interface generation process allows the specification of different tasks according to geometrically defined *Interaction Zones*. By acquiring the user's current position from sensors and comparing it to the geometric extents of these zones, the SmartMote is able to adapt its user interface consistently. This can lead to the exclusion of tasks that are not accessible from the user's current position. The following scenario illustrates the usefulness of location-dependent adaptation:

> Kai is an employee in the *SmartFactory*$^{\mathbf{KL}}$. After a week of restful vacation, he comes back to his working place. At the entrance, he picks up the SmartMote and enters his credentials. The SmartMote recognizes him and builds its user interface according to the preferences in his user profile. It seems that during his absence a module has been replaced by a newer model. The SmartMote highlights the new device in its navigation bar. Kai confirms the notification and begins his normal work routine. After some time, the last production order finishes and there are no further orders scheduled for today. Curious about the new device Kai decides to inspect it more closely. He activates the SmartMote's navigation routine and is guided to its location. The new module has already been equipped with an NFC tag, so Kai touches the marked point with the SmartMote and instantly the device offers its full functionality in the user interface. Impressed by the new device's capabilities he decides to test a newly added feature. After refilling an empty feed tray, he puts in an order for a few test samples. Unfortunately, the SmartMote does not allow him to activate the commissioning module. Instead, a notification informs him that he is standing in the safety zone of the module's automatic robot arm. After he clears the area, the SmartMote allows the start of the production. Satisfied with the samples Kai walks into another room of the *SmartFactory*$^{\mathbf{KL}}$ to have a chat with his colleagues. As he leaves, the SmartMote deactivates most control elements and shows only a summary of the most critical parameters.

The scenario shows how a universal control device in an ambient intelligent, user-centered environment can assist humans in their workflows. The SmartMote therefore not only provides the *anywhere, anytime* computing, but also follows another principle of ambient intelligence: Providing "the right thing at the right time in the right way" [HWM^{+}03].

2. Application Domain

This chapter gives an overview over the application domain. First, Section 2.1 introduces the term and characteristics of ambient intelligent production environments. Section 2.2 describes the *SmartFactory*$^{\mathbf{KL}}$, which understands itself as the first ambient intelligent factory worldwide. Model-based user interface development and two approaches to model the underlying tasks are explained in Section 2.3. Finally, Section 2.4 shows the practical application of these technologies in the SmartMote prototype.

2.1. Ambient Intelligence in Production Environments

The paradigm of *ubiquitous computing* was introduced by Mark Weiser in the early 90s. In his vision, users are permanently surrounded by computers that integrate seamlessly in everyday life. This constant background presence supports users' activities without them being actively aware of it. By pushing the technical aspects of computer usage to the background, users can focus on their practical use. Such machines that fit into human environments will raise the quality of life significantly [Wei91].

One step in Weiser's direction is the concept of *Ambient Intelligence*. The ISTAG vision states that humans will live in *Ambient Intelligent Environments*, "surrounded by intelligent interfaces supported by computing and networking technology that is embedded in everyday objects such as furniture, clothes, vehicles, roads and smart materials - even particles of decorative substances like paint" [RVDA05]. Such an environment should be aware of human presence and personalities and adapt its behavior according to users' needs, thereby constantly learning from their responses [RAS08]. The purpose to serve human needs drives the design of objects in ambient intelligent environments. Aarts and Marzano identified five key characteristics [AM03]:

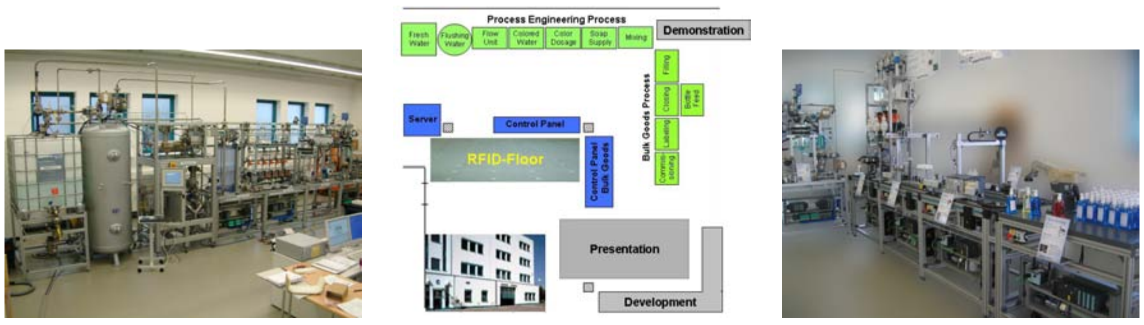
- *Embedded*: Similar to ubiquitous computing, devices are seamlessly integrated into objects. Technology is moved to the background.
- *Context-aware*: The devices are aware of persons and the current situation.
- *Personalized*: Not only do the devices support humans in general. They can be customized to the special needs of specific persons.
- *Adaptive*: They change according to human responses.
- *Anticipatory*: The devices anticipate human needs without being explicitly told.

Of special interest in this thesis is the second point. Knowledge about the current surroundings can be used to filter available interaction options. While this may seem contradictory at first, removing non-viable options greatly reduces the complexity of human-computer interaction. Easy-to-operate devices are more likely to be accepted and therefore support the human-centered approach as described in the *embedded* characteristic. Context information in its broadest definition (cf. Section 4.1.2) is also a basis for the personalization, adaptation and the anticipation of user needs.

The possible application areas of ambient intelligence are as diverse as today's environments. Research is done on many topics including *Smart Homes*, that explore possibilities for future living spaces, *Ambient Assisted Living* with the goal of assisting elderly people in their home environment, *Ambient Assisted Working*, which brings ambient intelligence to workplaces, and many more. The focus of this thesis is on *Ambient Intelligent Production Environments*. While no such environment exists in practice (yet), the next section introduces the *SmartFactory*$^{\mathbf{KL}}$ as test bed and demonstration facility.

2.2. *SmartFactory*$^{\mathbf{KL}}$

The *SmartFactory*$^{\mathbf{KL}}$ is the first test bed for ambient intelligent production environments worldwide. Funded by the non-profit registered association "Technology Initiative SmartFactory KL", it has provided a platform for the demonstration and development of new technologies since 2005. Members of the initiative represent different sectors from industry and research, including producers of factory equipment [ZÖ9]. The facility itself is located at Siegelbach / Kaiserslautern[1] and operated by the German Research Center for Artificial Intelligence (DFKI) [Deu16]. The demonstration area consists of a process engineering and a bulk goods process (cf. Figure 2.1). Devices from various fabricators are used to demonstrate a procedural process for the automatic bottling of liquid soap [ZÖ9].



(a) View on the process engineering process [Tec16]

(b) Floor plan [ZÖ9]

(c) The bulk goods process [Tec16]

Figure 2.1.: The *SmartFactory*$^{\mathbf{KL}}$. The demonstration area shows the liquid soap bottling process.

[1]It will be moved to a site near the Technical University Kaiserslautern in 2011.

The vision of the *SmartFactory*$^{\mathbf{KL}}$ is to provide a plant which is

- arbitrarily modifiable and expandable (flexible),
- connecting components from multiple manufacturers (distributed, networked),
- enabling its components to perform context-related tasks autonomously (self-organizing) and
- emphasizing user-friendliness (user-oriented). [Tec16, BGM+09]

A number of technologies support these goals. Wireless communication systems like WLAN, Bluetooth, ZigBee, NFC and RFID are deployed in the facility. The permanent WLAN connection enables mobile control of the devices in the piece goods part. Of special interest in this thesis are the available positioning systems. Large parts of the factory floor are prepared with an RFID grid. As the RFID tags are immovable and their position is known, a movable object with an RFID reader can derive its position from the latest read RFID tag. The same goes for a number of NFC tags which are mounted at the front of important modules. A commercial three-dimensional positioning system based on ultrasonic waves is also deployed and enables the retrieval of tags' coordinates over WLAN via a webservice.

2.3. Model-based User Interface Development

Modeling is the "abstraction of a real system by removing the irrelevant details in the current level of abstraction" [WFRS07, BRJ05]. It is apparent from this definition that a model does not describe all attributes of real-world objects. Omitting unnecessary details in the abstraction reduces the complexity of the description [DKM18]. As models are based on a formal syntax, they can be checked for consistency and the absence of errors. This can be done automatically by software tools. Finally, transformations are possible. Different models containing information on the same real-world entity can be combined to generate a single model with richer content. Models with rich content can be further abstracted or restructured. All of these operations can be performed automatically, which eliminates the possibility of human errors during the process.

Model-based User Interface Development (MBUID) utilizes these advantages. Multiple formal models covering different aspects of the developed user interface (UI) are used to (semi-) automatically generate the final UI. This is done by a series of model transformations where abstract models are gradually transformed into more concrete models. Semi-Automatically means that the transformations can be done by experts using respective software tools at design-time or by a run-time process. MBUID provides several benefits. Abstracting from a concrete implementation facilitates the maintaining of different versions of an application across multiple devices [CCT+03]. As additional effects, the UI can be verified and validated and the consistency to older versions is maintained [CLC04].

Various MBUID systems use different models as a basis for the generation. While there is no consensus on the final composition of the so-called *Interface Model* [Pue97], it is commonly accepted that the core consists of at least the *task, dialog* and *presentation* models [LC04]. Dialog and presentation model are *concrete* models, which means that their elements can be directly mapped to elements in the UI. While the dialog model describes the interaction between the user and the UI, the presentation model contains information about the visual display of components. The task model is an *abstract* model as its elements do not directly correspond to GUI elements. A task model describes "the tasks (and sub-tasks) a user has to execute to achieve a certain goal" [WFRS07]. Tasks are hierarchically structured and comprise the activities necessary to fulfill the task.

The core interface model is sufficient for generating a functional static UI. However, the requirements in ambient intelligent environment demand more. The vision states that interfaces should adapt to the current situation. Hence, an additional model is required: the *context model*, which contains information about external factors influencing the interaction. Figure 2.2 shows a simplified version of the Cameleon reference framework [CCT+03].

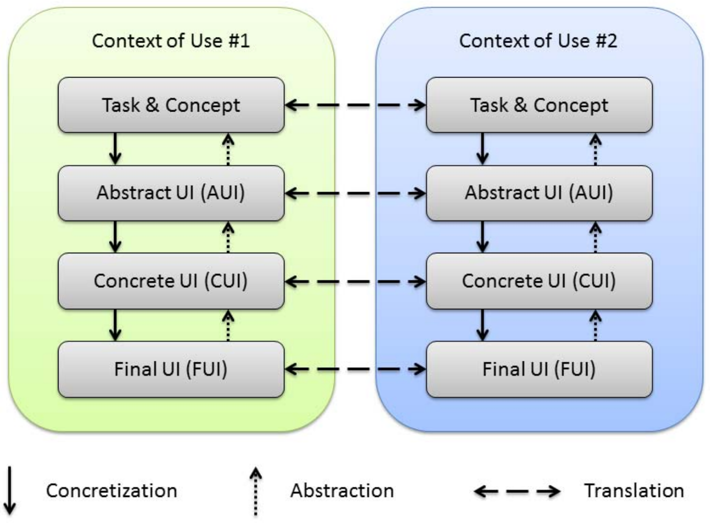


Figure 2.2.: Four levels of the simplified Cameleon framework [LVM+05]. The models are translated according to the current context of use.

Each of the previously mentioned models is influenced by the current context of use. If the circumstances change, the models need to be translated according to the new context of use. This means that information about considered contexts has to be included in the models during their development. They have to be *context-sensitive*. This enables a run-time process to derive a *context-insensitive* representation for each individual context of use. While each individual model may be context-sensitive, it suffices to embed context knowledge in the task model. The final steps of an MBUID process including different contexts are shown in Figure 2.3. Task models for every context of use are generated from the general task model. These individual models are activated during run-time if the respective context applies.

Pribenau et al. present three approaches for the connection between context-sensitive and context-insensitive parts of a task model [PLV01]:

- In a *monolithic approach* both parts are included in one task model.

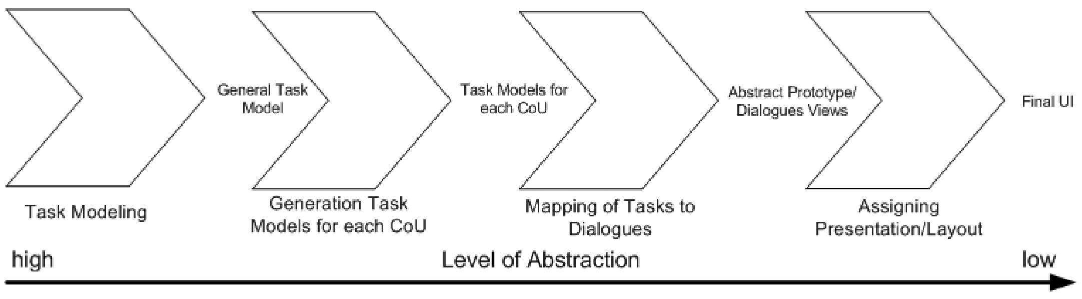

General Task Model
Task Models for each CoU
Abstract Prototype/ Dialogues Views
Final UI
Task Modeling
Generation Task Models for each CoU
Mapping of Tasks to Dialogues
Assigning Presentation/Layout
high
Level of Abstraction
low

Figure 2.3.: MBUID process including consideration of multiple contexts [WFRS07]

- The *graph-oriented approach* separates the two parts. Logical mapping between them is done by general connections.
- The *separation approach* also splits the model into two parts, but the connections between them are specialized and take the form of a decision tree.

The following sections introduce two notations for context-sensitive task modeling.

2.3.1. ConcurTaskTrees

The *ConcurTaskTree* notation (CTT) was introduced by Paterno et al. in 1997 [PMM97]. It is the most commonly used task modeling approach in various domains. CTT uses a graphical, tree-like representation in which nodes represent (sub-) tasks. The trees' vertical levels determine the current level of abstraction. A parent node is either decomposed or refined by its children. The horizontal order defines temporal relations between tasks on the same level. Horizontal dependencies control concurrency, synchronization, iteration and information passing between neighboring nodes. CTT distinguishes four types of tasks:

- *User tasks* are executed solely by the user. Selecting a favorite dish from a menu is an example. Data may be supplied by the system and the user may input a response.
- *Application tasks* are initiated and executed by the system without user interaction.
- *Interaction tasks* are activated by the user and resolved through user interaction with the system.
- *Abstract tasks* are complex tasks that fall in none of the above categories.

Figure 2.4 shows an example CTT for checking the temperature using different scales. The user selects the thermometer (e.g. in a graphical interface) and has the choice to select between three different scales. The system converts the temperature to that scale and displays it for the user to read.

The original CTT depicts a context-insensitive task modeling approach. Luyten, Clerckx and Coninx proposed an extension that follows the graphical approach, but was inspired by the decision trees used in the separation approach [CLC05, LC04]. A new type of node, the *Decision Node*, is introduced. Decision nodes contain boolean expressions operating on context data. During run-time the nodes are evaluated to *true* or *false*. For both cases connections to either context-insensitive sub-trees or another decision node are defined. The decision node is replaced by the root of the correct sub-tree or by the next decision node. After all decision nodes have been resolved, the tree depicts a normal CTT for the current context of use.

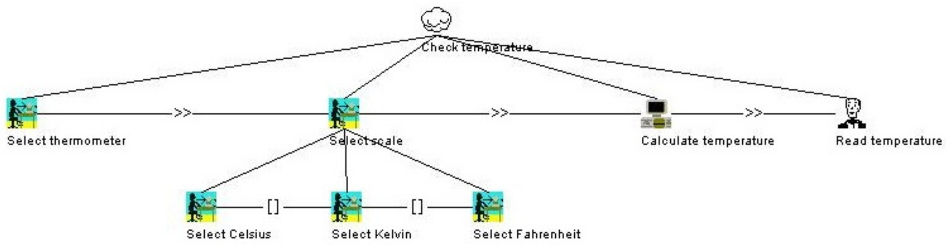


Figure 2.4.: Example in CTT: Check temperature in different scales

2.3.2. Room-Based Use Model

The *Useware*[2] *Markup Language* (useML) was originally designed for the systematic development of interactive devices [Reu03]. It is an XML-based, hierarchical approach in which a *Use Model* is defined by multiple *Use Objects*. Use objects in turn are a composition of *Elementary Use Objects* of which five different kinds are defined: *change, trigger, select, enter* and *inform* (cf. Figure 2.5).

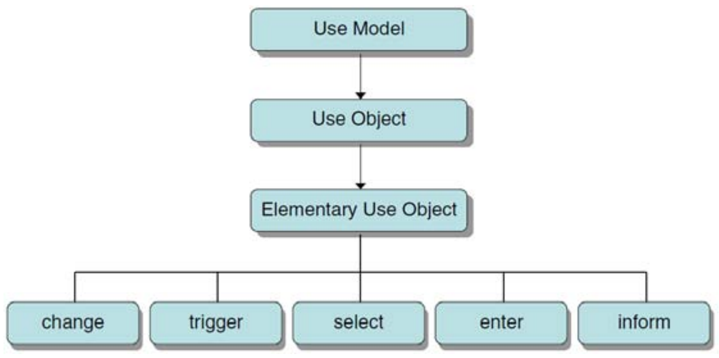


Figure 2.5.: Original Use Model structure [BGM+09]

The original specification has been extended multiple times to include temporal relations, user roles and personas [MG08]. The third-generation UseML has made the biggest contribution towards context-sensitivity. Görlich provided the possibility to model spatial relations. The result is called *Room-Based Use Model* (RUM) [Gö09]. Figure 2.6 shows the overall structure. *Organizational Rooms* may contain other organizational rooms (based on spatial or logical

[2] *Useware* is a collective term for all hardware and software components that are required for the use of a system [Use19]

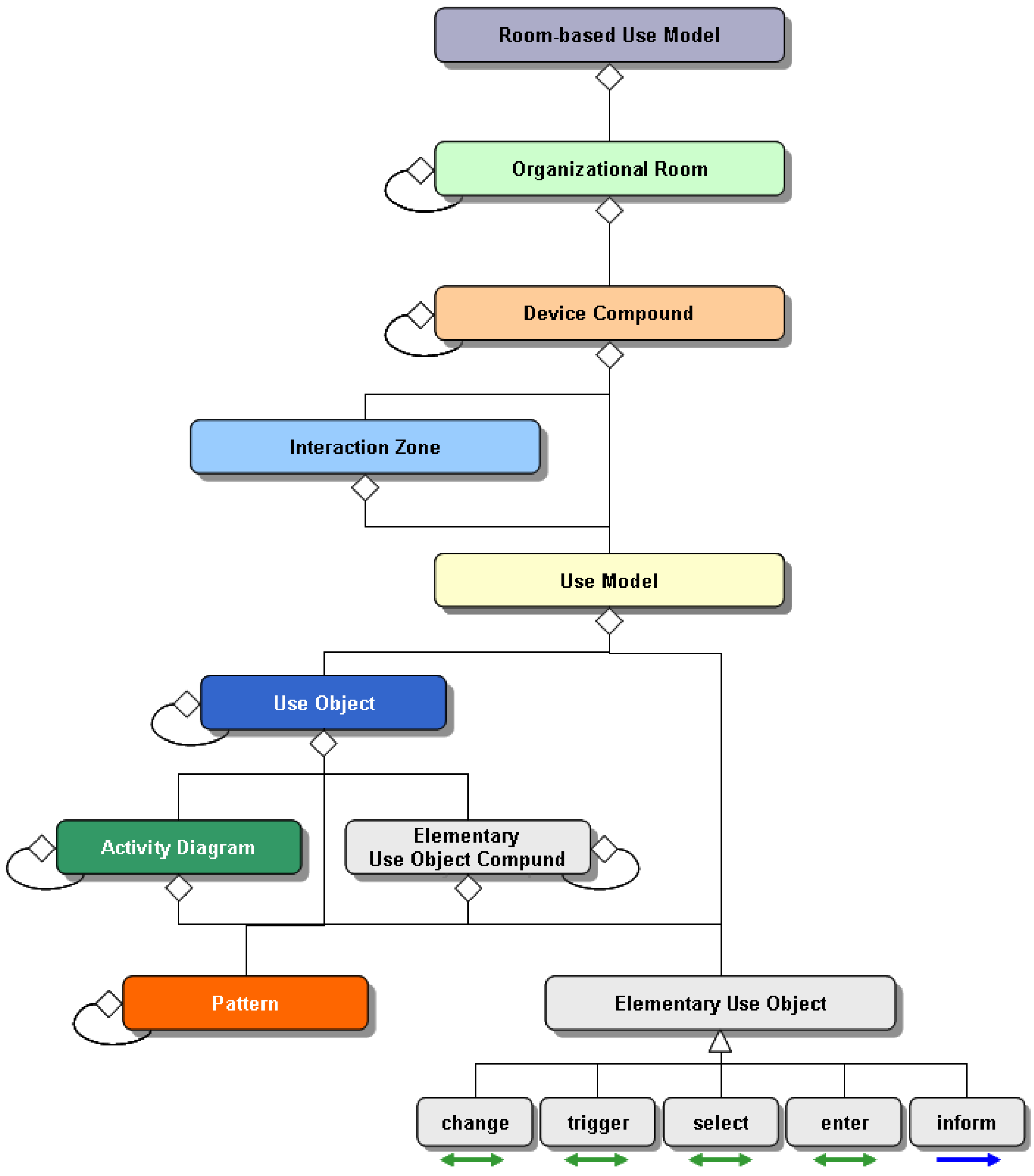


Figure 2.6.: The Room-Based Use Model. Device Compounds may include Interaction Zones to model different behavior depending on the user's position [BGM+09]

containment) and device compounds. The point of interest in this thesis is the possibility to specify *Interaction Zones*. This concept was originally used for informal communication in smart working environments (cf. [SRP$^+$03] and Section 4.2.3) and enables the specification of different use models depending on the user's position.

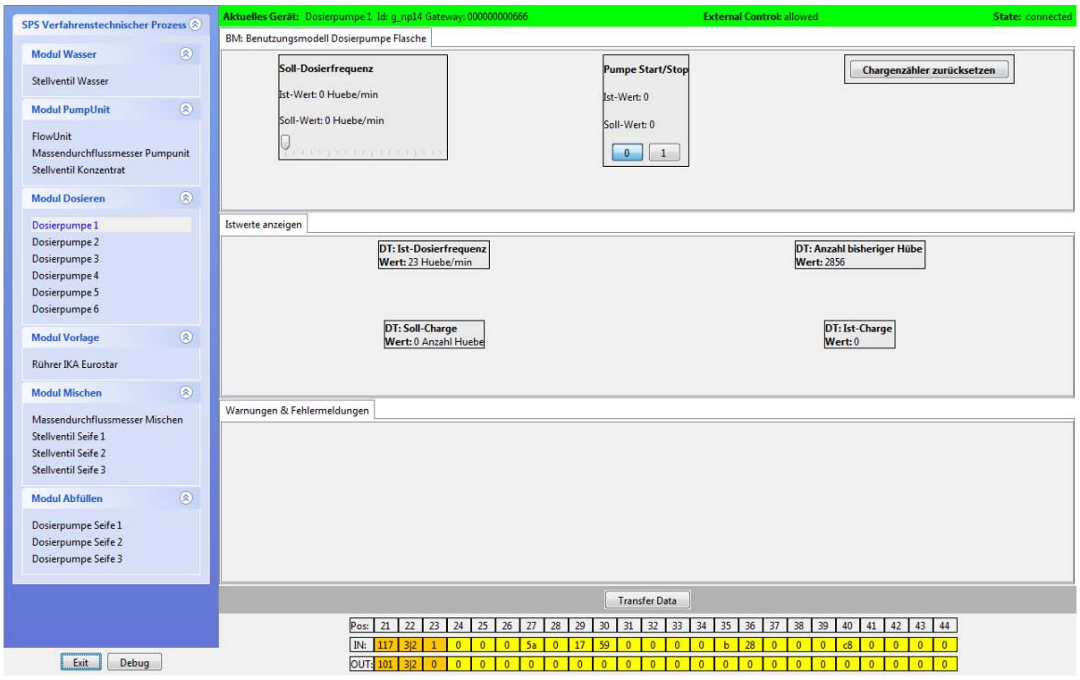
The advantage of useML over CTT is the elementary use objects' level of detail. The interaction between the user and the system can be described on a fundamental level in useML, which is more suitable for a consistent MBUID process [MG08]. The downside of useML is the integration of many extensions directly into the model. It follows the monolithic approach, which can lead to large and complex models.

2.4. SmartMote

The SmartMote is a universal control device for the *SmartFactory*$^{\mathbf{KL}}$. It uses an MBUID process to handle the complexity of ambient intelligent production environments. Based on a RUM of the *SmartFactory*$^{\mathbf{KL}}$, a run-time process generates the user interface. The principle of *Universal Control* is used to handle the complexity stemming from differences in the interaction concepts of devices from multiple manufacturers. The user is presented a holistic view of his environment and can control all devices from a single, mobile TabletPC. The mobile communication is thereby handled via bluetooth. A *Function Model* integrated into the RUM provides communication information on two levels [BMGM09]: On device level, the model describes the initiation and structure of the device's communication protocol. The elementary use objects contain links to this information, which finally activate the communication.

Figure 2.7 shows the generated user interface. Device compounds and devices are contained in the navigation bar on the left side of the screen. Selecting a device triggers its use model to be shown on the right. Currently, each elementary use object is mapped to one widget in the graphical user interface (GUI). The GUI is functional and can be used to control the *SmartFactory*$^{\mathbf{KL}}$. Improvement of the relatively simple visual appearance is part of ongoing research. Steps towards the integration of usability patterns are already taken (cf. Figure 2.8) [SKD$^+$10].

The SmartMote is already context-aware in the sense that it is able to adapt to changes in the environment. The task model file is continuously polled for changes. The provision of a new model file, for example via an USB stick, is possible during run-time. The new model is then automatically detected and its changes are reflected in the UI. The detection of malfunctioning or non-present devices with the help of the function model is also implemented. Devices that are specified with a valid function model in the RUM, but cease to answer to bluetooth requests are removed from the navigation bar. The adaptation can occur in three modes: *ad-hoc*, by *notification* or by *confirmation*. A recently conducted pilot study favors the confirmation approach, but only the final study will show definite results [BGSG10]. All adaptation mechanisms aim at keeping the user interface consistent to the real environment, always focusing on the user's needs and abilities.



Aktuelles Gerät: Dosierpumpe 1 Id: g_np14 Gateway: 000000000666 **External Control:** allowed **State:** connected

Soll-Dosierfrequenz
Ist-Wert: 0 Huebe/min
Soll-Wert: 0 Huebe/min

Pumpe Start/Stop
Ist-Wert: 0
Soll-Wert: 0

DT: Ist-Dosierfrequenz
Wert: 23 Huebe/min

DT: Anzahl bisheriger Hübe
Wert: 2856

DT: Soll-Charge
Wert: 0 Anzahl Huebe

DT: Ist-Charge
Wert: 0

Pos:	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44
IN:	117	3|2	1	0	0	0	5a	0	17	59	0	0	0	0	b	28	0	0	0	c8	0	0	0	0
OUT:	101	3|2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 2.7.: Current SmartMote GUI with debug information at the bottom

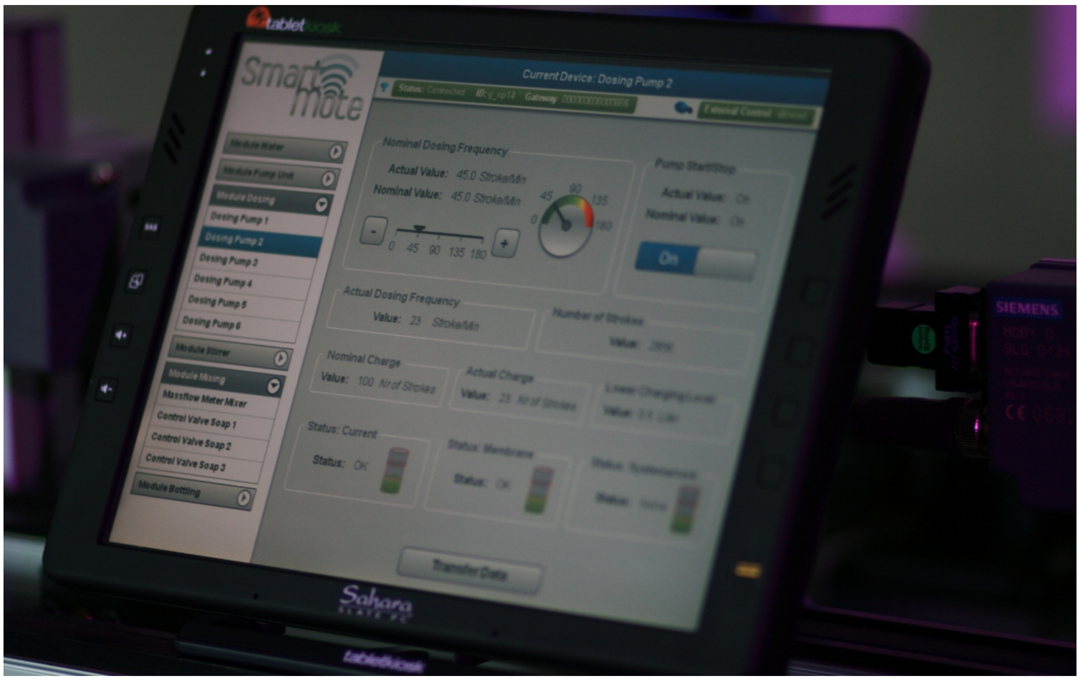


Figure 2.8.: SmartMote GUI with integrated use of usability patterns

3. Goals and Course of Action

Goals

The automatic MBUID generation process of the SmartMote is based on the *Room-Based Use Model* (RUM), which allows the specification of *Interaction Zones*. Figure 3.1 shows the part of interest. *Device Compounds* may be an aggregation of *Interaction Zones* with different *Use Models*. Which use model is currently active in the device compound is determined by the location of the user.

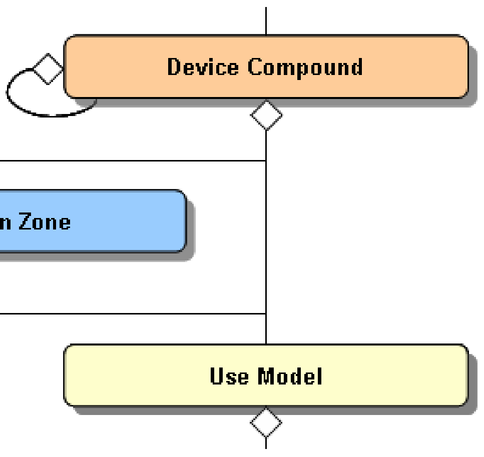


Figure 3.1.: Detail of the RUM specification. *Device Compounds* may be a aggregation of *Interaction Zones*

The goal of this work is to develop the infrastructure necessary to sense, store and provide information about the user's location. Three aspects must be considered:

- A *context model* has to be developed that stores geometric information about the user, objects in the environment and the zones as they are only modeled in a symbolic way in the RUM.
- Existing *sensor systems* must be integrated which sense the current user position and provide them to the model.
- A *reasoner component* has to be developed, which derives the user's position from incoming sensor data, determines the currently active zones (i.e. the zones containing the current user position) and filters the RUM accordingly.

The developed system should also be extensible in a way that allows the integration of additional context types. In general, the system should provide answers to the following questions:

- What is the current position of the user and the SmartMote?
- Where was the SmartMote during the last n seconds?
- Which interaction zones are currently active?

Structure

The remainder of this thesis is structured as follows. Chapter 4 examines the general *notion* of context, context *definitions*, its *uses* and how context can be *classified*. Existing approaches to context modeling in general and with a focus on location models are examined in Chapter 5. A discussion of possibilities to *manage* the context and frameworks with the ability to support multiple context-aware applications is also included in this chapter. The results from the analysis are used as a basis for the definition of a *formal context model* in Chapter 6. The developed *context engine* for the SmartMote and detailed implementation information are described in Chapter 7. The *feasibility* of the developed system is demonstrated in Chapter 8, which describes the test of the modified SmartMote in the *SmartFactory*$^{\mathbf{KL}}$. Finally, a *summary* and prospect for *future work* is given in Chapter 9.

4. Context of Use

In this chapter the *notion* of context is formally introduced. Both the origin of context and its transfer to the computer science domain are discussed in Section 4.1. Some *examples* of context-aware applications are given in Section 4.2. These include context in their application logic, but do not model it explicitly or embed it in an MBUID process. Finally, some context *classification* attempts are provided in Section 4.3 which also motivates the use of context *reasoners*. Section 4.4 *summarizes* the results.

4.1. Definitions

During the research of context-awareness authors have come up with a number of different definitions. While most of them have a common basis, they also differ in many details. A clear definition of the terms used in this thesis is mandatory. The word "context" itself has meaning in natural language and has been transfered to the computer science domain. This section introduces important definitions and gives a brief historical review over their development.

4.1.1. Context in Natural Language

Context in natural language is used to refer to the circumstances of an entity or in which an event happens. Furthermore, the semantic meaning of a sentence may only become clear from its context. Synonyms for *context* in natural language are *situation*, *background*, *setting* or *surroundings*. The first definition given by the Merriam-Webster online dictionary summarizes:

> "The parts of a discourse that surround a word or passage and can throw light on its meaning"
> [Mer07]

Context is necessary to choose the correct meaning of an expression. The word 'state' may describe a political community with a government or the condition of an object such as solid, liquid and vapor. Different domains introduce additional meanings. The medical state describes a patients vital functions whereas the state of an object in computer science describes the values of its internal variables at a specific time. Context can therefore be seen as a filter on the set of possible meanings.

4.1.2. Context in Ambient Intelligent Environments

The second and more general definition of context in the Merriam-Webster online dictionary also constitutes the linked nature of context information:

> “The interrelated conditions in which something exists or occurs”
> [Mer07]

The complexity of context makes it hard to give a clear definition. The term has been defined from different perspectives and to different extents during the last two decades. Initially, many researchers tried to grasp the nature of context by enumerating properties of users’ or applications’ environments that seemed relevant. The probably earliest definition was given by Schilit, Adams and Want:

> “Three important aspects of context are: where you are, who you are with, and what resources are nearby. [...] Context includes lighting, noise level, network connectivity, communication costs, communication bandwidth. and even the social situation.”
> [SAW94].

Chen et al. extended this enumeration to also include the *Time Context*, such as time of day, week, month and season of the year [CK00]. Brown et al. refer to context as “location, time of day, season of the year, temperature, and so forth” [BBC97]. All of these definitions include important aspects of context, but are still limited to listing the notions. Later definitions try to catch context on a more abstract level. Dey et al. gave a more general definition that is widely accepted and comprehensive:

> “Context is any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves.”
> [ADB$^+$99]

This understanding of context as practically any information from an application’s or user’s environment is slightly constrained in the definition by Wurdel et al.:

> “The context of use is any information that can be used to characterize the situation of the environment, the user and the device of a software system, which is regarded as relevant for the interaction of the system and the user.”
> [WFRS07]

Because of its generality and applicability, the rest of this work understands context according to Dey and Abowd’s definition, but focuses on information relevant to the interaction between user and context-aware system.

4.1.3. Context-awareness

After discussing various approaches to defining context, the concept of *context-awareness* can be introduced. In general, an application or system is called *context-aware* if it *adapts* its behavior to changes in context. Sometimes, the terms *context-sensitive*, *context-enabled*, *context-aware computing* or *situation-aware* are used synonymously. Consistent to their understanding of context Schilit, Adams and Want define context-aware systems:

> “Such contex-aware systems adapt according to the location of use, the collection of nearby people, hosts, and accessible devices as well as to changes to such things over time.”
> [SAW94]

This and other previous definitions were analyzed by Dey and Abowd [ADB$^+$99]. They distinguish between two categories: *using* context and *adapting to* context. The definition of Salber et al. falls into the first category. They define the aim of context-aware computing as “to provide maximal flexibility of a computational service based on a real-time sensing of [...] context” [SDA98]. Definitions from the second category are more dynamic. Brown describes context-aware devices as devices that “automatically provide information and/or take actions according to the user’s present context” [Bro98]. Based on their study of these early definitions, Dey and Abowd provide their understanding of context-awareness:

> “A system is context-aware if it uses context to provide relevant information and/or services to the user, where relevancy depends on the user’s task.”
> [ADB$^+$99]

4.2. Examples of Context-aware Applications

This section introduces some examples of context-aware applications. The examples demonstrate how and which aspects of context have been used in working environments, for memory assistance and in information systems. This list is, of course, not complete, but gives an exemplary overview over context-aware applications. Classification schemes for the notions used of context are introduced in Section 4.3.

4.2.1. Working environment

Context-aware applications can assist employees in their everyday work. Context can be used to detect resources. For example, a print command issued from a mobile device, could automatically be forwarded to the nearest printer. Considering social situations is another possibility. Calls could be rejected or directed to an answering machine if the callee is currently in a meeting. *ActiveBadge* and *PARCTab* are two pioneer systems that were introduced in the early 90s. They were the first location-aware systems for working environments.

ActiveBadge

The *ActiveBadge* system [WHFaG92] was developed by *Olivetti Research Ltd.* which was acquired by AT&T in 1999 and became the *AT&T Laboratories Cambridge*. It is the first commercial system for tracking user locations. Employees submit their location by wearing a badge visibly at their body. Figure 4.1 shows the four generations of active badges. As the prefix *active* implies, the badges are periodically sending out a unique infra-red (IR) signal. A network of IR-receivers is able to receive these signals. A central server polls the receivers for “sightings” of the badges and maintains a central register of badge locations. It is therefore possible to track users with a temporal resolution of 15 seconds. The badges require batteries that last

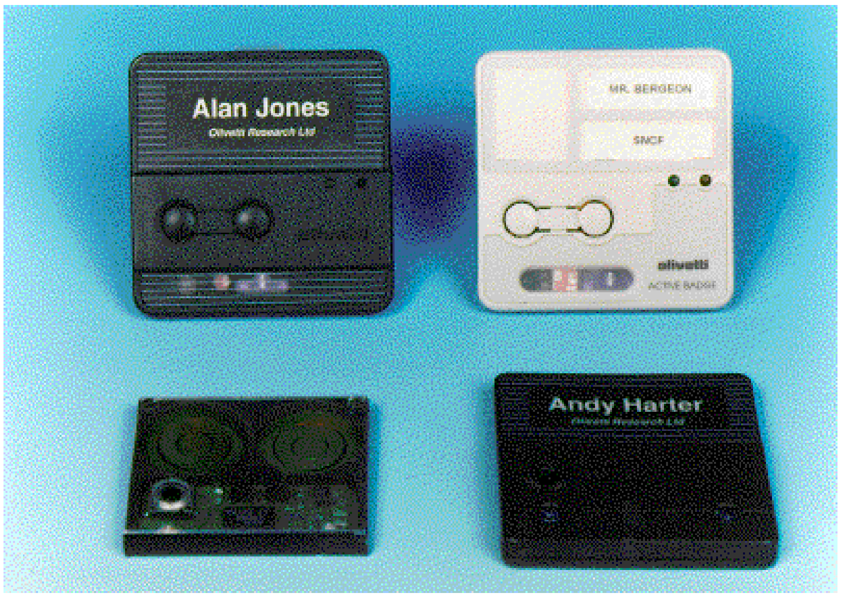


Figure 4.1.: ActiveBadge generations from the AT&T Laboratories Cambridge Archive [Ame13]

about one year. The system is very user-friendly as IR-signals are reflected by walls and other obstacles, so that users can be tracked even if there is no direct line of sight between the badge and the next sensor (e.g. when the badge is worn at the belt and the user is sitting at a table). The initial application of the system was designed as an aid for telephone receptionists. The receptionist could display a list of all staff members, their last known location and a probability value indicating the certainty of the information. This field showed a percentage below 100% if the user was moving. After five minutes without detection it switched to the time of the last sighting, after 24 hours to the last day and after one week of absence to "AWAY". Furthermore, it was possible to query the system for a specific location, other badges in the vicinity of a target and the history of a single badge. The system was later extended to automatically forward calls to the nearest phone of the employee. Another application was the automatic unlocking of doors. People stopped wearing the badges with the upcoming of mobile phones. It is interesting to know that the badges themselves make use of context data. They have a light-dependent component that can turn the IR-signal off to conserve battery power.

PARCTab

Researchers at the *Xerox Palo Alto Research Center* (PARC) developed the *PARCTab* system, which is based on mobile personal digital assistants (PDA) that communicates via infrared data-packets to a network of IR-sensors [SAG$^+$93]. The primary goal is to shift the execution of resource-intensive programs to remote computers. The PARCTab therefore works as a terminal station. The necessary continuous connection to the IR-network is facilitated by the possibility to seamlessly switch cells, i.e. taking the PDA from one room to another does not break the connection. The PDA itself has three different buttons and a touch sensitive display with a resolution of 128x64 pixels (cf. Figure 4.2). Applications that can be executed from the PARCTab include a dictionary, calendar and weather forecasts acquired from the internet. While the main goal is to facilitate mobility, the PARCTab also includes context-aware behavior. It is possible to forward an active workstation session to the nearest available computer. A collaborative drawing board for multiple PARCTabs in one room has also been realized [SAW94]. Finally, access to *virtual* objects is implemented. Files in the UNIX file system are associated with rooms. Users can deposit files or notes in a specific room that are presented to others who access the file system in the same room at a later time.

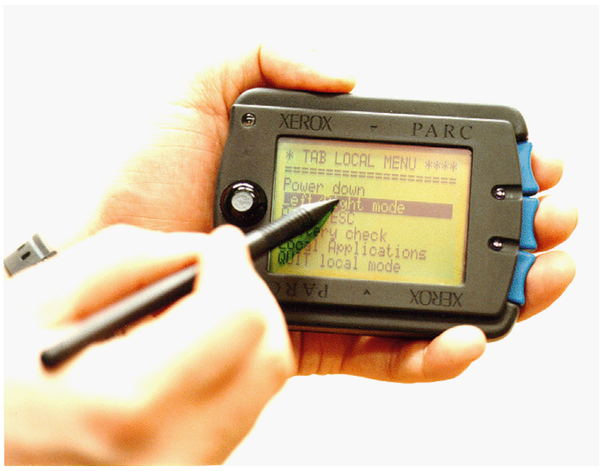


Figure 4.2.: The Xerox PARCTab [Xer13]

4.2.2. Memory Assistance

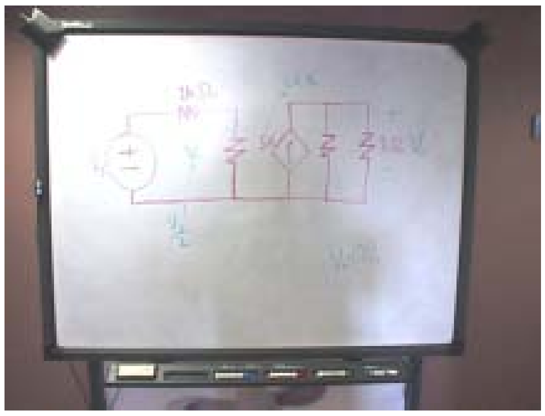
Memory assistance systems help users to remember things they did not consider important at the time of occurrence or supply information that is important in the current situation. They capture facts, notes, decisions, conversations and more types of data from one or multiple persons and provide them at a later time. Context can help to filter and sort the saved information according to the user's needs in his current situation. Context can also be used as a trigger for the capturing.

Stick-e-Notes

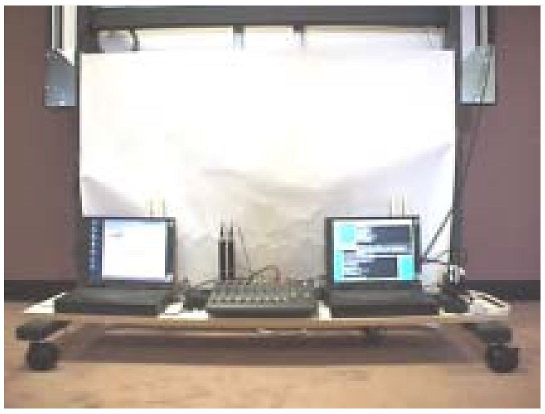
Brown et al. introduced what they call a *stick-e-note* [BBC97]. It follows the Post-it-note metaphor. Post-it-notes are used to write short pieces of information and stick them to your monitor or any other place where you frequently see them and thus are reminded of their content. They are also used to deposit short messages for other people. For example, a note saying "back in five minutes" on your door informs visitors of your status. In stick-e-note terms, this would mean that you define the content (the text "back in five minutes") of a note, add the target context ("location: my door, situation: I am absent") and save it as stick-e-note. Any visitor, whose context matches the note's context, is then shown the content. This approach can also be used to record a history of notes for later review. For example, you could record information and pictures of a touristic tour as stick-e-notes and review the whole trip at a later time or even publish the complete history as a website. The system originally used mobile devices like PDAs with GPS coordinates and a relatively limited set of context information, but was later extended into a framework to support context-aware applications [Bro96].

DUMMBO

The *Dynamic Ubiquitous Mobile Meeting Board* (DUMMBO) was built at the *Georgia Institute of Technology*. Brotherton et al. wanted to create a prototype for the capturing of spontaneous, unplanned and informal activities that take place around a whiteboard [BAT98]. Their argumentation is that people involved in an activity often neglect the recording of events. Participants may not pay full attention, underestimate the importance of an event, may be too engaged in the activity or simply may be not equipped with the means to take notes because of the spontaneous nature of the activity. In research labs or lecture rooms, it is not untypical to spontaneously engage in a discussion about the latest project. Sketches and notes are often made on a nearby whiteboard. DUMMBO is able to digitize what is written on its surface. Two microphones at the top edges are able to record speech in front of the board. Figure 4.3 shows the front and back of the board.



(a) Front view of DUMMBO. Aside from the markers no extra controls are required.



(b) Audio mixer and recording equipment at the back. The setup can be rolled from one location to another.

Figure 4.3.: The DUMMBO setup

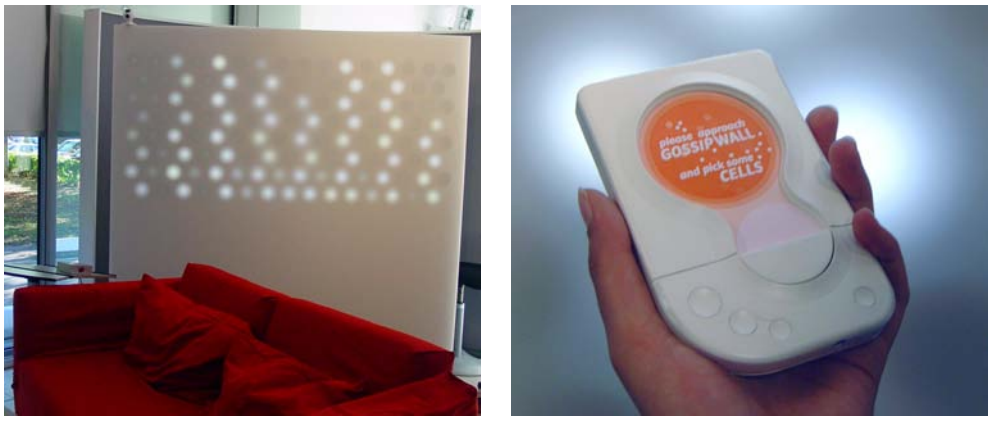
The whole setup is mobile and can be rolled from one location to another. Context information is used as a trigger and filter. To start a session, users simply have to draw something on the board. From this point on, everything that happenes on the board as well as everything spoken in front of it is recorded. A session ends if the noise level stays below a threshold for a fixed amount of time. The second use of context applies when browsing the list of recorded sessions. It can be filtered by time, length and participating members. The board has a web-interface for playing back the sessions and showing what the board looked like at a specific time during the session.

4.2.3. Ambient Information Systems

Ambient information systems are pieces of calm technology [WB95]. Their main purpose is to display non-critical information without disrupting the user. Email-notifications, for example, usually do not require immediate attention by the user. Displaying a pop-up window whenever a new email arrives might disturb the user's workflow, because his attention is drawn away from his current task. Mankoff et al. define ambient displays as "abstract and aesthetic peripheral displays portraying non-critical information on the periphery of a user's attention" [MDH+03]. Some of these devices just unobtrusively display context information like the weather conditions outside a building or the latest stock prices, but many are also context-aware themselves. The displayed information and notification level can adapt according to changes in the environment [TLG07]. The following two examples show, how awareness of location and user interest can influence the display of information.

Hello.Wall

The Hello.Wall system [PRS+03,SRP+03] was developed as part of the *Ambient Agoras* project [PSR+04]. The wall consists of 124 cells, each containing an LED cluster and a short-range transponder. The wall can show different light patterns, depending on the identity and distance of people passing by (cf. Figure 4.4(a)). The authors consider it to be an atmospheric influence in work environments. As the light patterns are personalized, the conveyed information has encoded character. While initiated members can read the information, visitors or other persons might just perceive it as a decorative element. The wall can also interact with mobile devices called "viewports" (cf. Figure 4.4(b)).



(a) Front-view of the Hello.Wall in ambient mode (b) The viewport used to interact with the wall

Figure 4.4.: The Hello.Wall system consisting of 124 LED cells and hand-held viewports [Fra14]

These hand-held devices can be used to interact with the wall. As each individual cell has a transponder they can be addressed individually. Cells are able to contain general data, personalized messages or they can trigger an application on the viewport. This process is called "borrowing" the viewport. For example, a company-wide survey where the viewport functioned as voting device was realized this way. To be able to detect nearby persons, RFID sensors are used. They cover two ranges and the authors therefore introduce three *zones of interaction* as shown in Figure 4.5. The *ambient zone* is defined as the space not covered by the sensors. People passing by without being detected, experience only generic light-patterns. The next zone is the *notification zone*. If an individual passes by at this distance, notifications are shown via light-patterns. Depending on the application, the user is able to interact with the wall at this distance using his viewport. The closest zone is the *interaction zone*. Users in this proximity are enabled to interact with the individual cells. While the interaction zones are based on user proximity in three stages, the concept can be generalized to consider arbitrary coordinates. The room-based user model from Section 2.3.2 makes use of this idea.

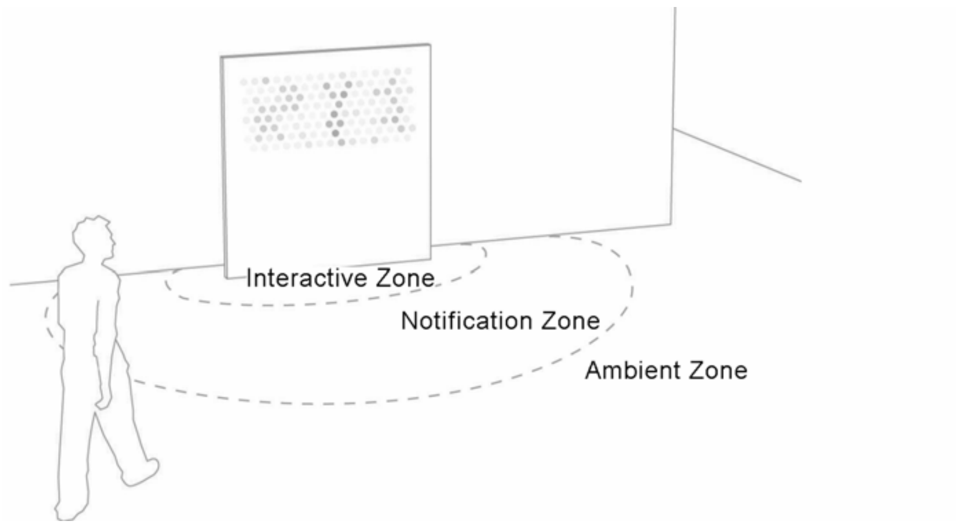
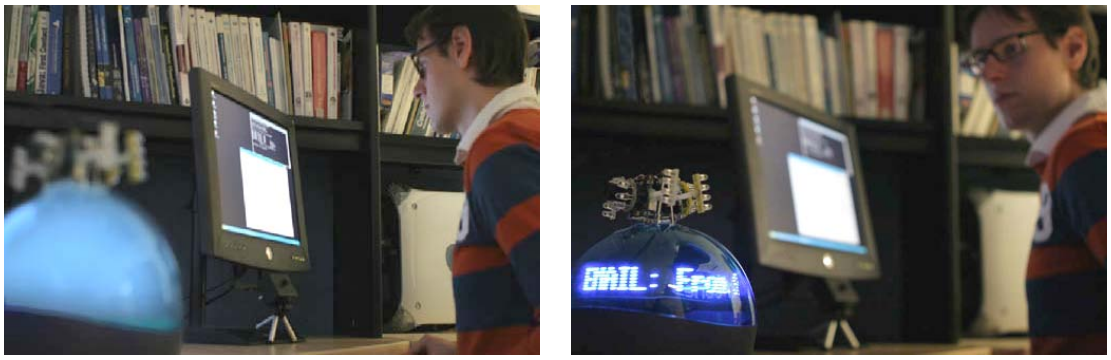


Figure 4.5.: Three interaction zones based on user proximity [SRP+03]

AuraOrb

AuraOrb is a notification system that uses context information to determine user interest in the appliance [AVSC06]. Altosaar et al. approach the problem of multiple systems trying to catch users' attention. In modern workplaces many different systems use a wide spectrum of means to draw user attention: A pop-up window on the computer screen notifies about new messages, a blinking icon in the task tray indicates instant messenger activity, an alarm on the mobile organizer reminds of the next meeting and at the same time the phone is ringing. In this enumeration only the phone call might require immediate action by the user. AuraOrb is envisioned to integrate all the non-critical notifications in one device and only display them, if the user initially expresses interest in the orb. The authors use eye contact as a primary context information to determine the object of current user interest. In the demonstration setup both the computer screen and AuraOrb were equipped with eye contact sensors. This enables the system to determine if the user is just working on the computer, looking at the orb or none of both. This influences AuraOrb's notification level. Three operating states are possible: ambient, semi-foreground and foreground. If no notification is available, the orb stays in ambient mode and is completely unobtrusive. When a notification arrives, the orb begins to emit an ambient glow (as shown in Figure 4.6(a)). Subsequent notifications trigger no further reaction, the glow is maintained. If the user decides to retrieve the notifications by looking at the orb, they are displayed in a ticker-like format on the orb's surface (Figure 4.6(b)). Similar to an answering machine. AuraOrb is also touch-sensitive. A single "touch" on the surface while it shows a message skips to the next one and a "double touch" opens the corresponding application with the message details on the user's workstation.



(a) AuraOrb signaling available notifications by an ambient glow

(b) AuraOrb with full user attention displaying the header text of a notification

Figure 4.6.: AuraOrb in the modes *semi-foreground* and *foreground* [AVSC06]

4.3. Classification of Context

In addition to the general definition of context in ambient intelligence, classifications can help to form an understanding of context. The examples given in the previous section show the variety of possible applications. Categorizations help identifying single types of context and its properties, both of which should be considered when developing a context-aware system. This section gives an overview of different categorization approaches that researchers have come up with.

4.3.1. Primary and Secondary Context

Dey and Abowd characterize context as two-tiered system [ADB+99]. In their opinion, the four context types *location*, *identity*, *time* and *activity* are *primary* context types. They are used more frequently and are thus more important than the other types. The authors characterize context-aware applications as systems that "look at the *who's*, *where's*, *when's* and *what's* (that is, what the user is doing) of entities and use this information to determine *why* the situation is occurring" [ADB+99]. Primary context may also be used to indicate second-level or *secondary* context types. For example, a given identity (a person) may be linked to many related attributes like height, eye color and friends. So, the primary context of one identity may be used to find secondary context of this entity and primary context of other entities (the list of friends contains other identities). A combination of first-level context may also be necessary to conclude a second-level context. The example Dey and Abowd use is a weather forecast, which requires both location and time. With the help of this classification, developers can structure context types in their work. This can help to identify additional context types, but also to filter out redundant secondary context types, so that only those remain which are relevant for the primary context.

4.3.2. High and Low Context Types

The report of Chen and Kotz [CK00] includes a differentiation between high and low context types. Low context types can be acquired directly by physical sensor hardware whereas high-level context is derived from the refinement and combination of low-level context. Low-level context is characterized by a simple structure and includes information like the current time of day as measured by an internal clock or the noise level registered by a microphone. The two authors make no distinction between sensors included in context-aware devices and remotely available sensors. Specifically, the identified low-level context types are:

Location Location can be sensed in many ways. Chen and Kotz distinguish two kinds of location sensing. The first requires the users to *actively* supply their location to the system. This can happen by a fingerprint sensor at the entrance to each room, sliding a badge at a sensor when entering or leaving or by monitoring on which workstation the user is logged in. This relies on the cooperation of the user and is only accurate if the user remembers to supply the information regularly. The second kind senses the location *passively* and mostly *continuously*. This usually requires the user to carry some kind of tracking device. Outdoors, the Global Positioning System (GPS) may be used. Typical for locating cellphones is the *Cell of Origin*-method which uses the user's base station as a reference point. Another possibility to sense the location is via *multilateration* (also known as *hyperbolic positioning*) which uses the *time difference of arrival* (TDOA) to calculate a position by measuring the flight-time of a signal to at least three different receivers with known positions. Location-sensing is always subject to inaccuracy.

Time Time is usually obtained by a built-in clock. In addition to the current time of day, the date, season of the year, etc. may be of interest.

Nearby objects If not only the location of the user but also of objects is recorded, a list of objects in direct vicinity can be determined and used.

Network bandwidth The available network bandwidth is important if many devices share a communication channel. In ambient intelligent environment this is frequently the case as wireless communication like WLAN and Bluetooth is limited to a fixed number of frequencies.

Orientation Many of today's mobile devices are equipped with orientation sensors. These can sense the orientation of the device in relation to the ground (e.g. to switch a display to landscape view if turned) or they sense the geographical direction of view (like a compass).

Other Additional low-level context includes sensors for lighting, vibration, temperature, sound, pressure and more.

High-level context builds on top of low-level context. By taking the data from multiple low-level sensors into account, more general facts like “current activity” or “user’s mood” can be deduced. This task is often carried out by a *reasoning process* on the model. The challenge is to overcome the inaccuracy and ambiguity of low-level data. While low-level context is structurally simple and continuous, high-level context tends to be complex and discrete. In many works this distinction is implicitly included. For example, Hofer et al. call the categories *physical* and *logical* [HWM+03].

With this distinction explained the process of *reasoning* can be introduced as “to automatically deduce further, previously implicit facts from explicitly given context information” [Ay07]. This process can be executed as a part of the internal application logic like in the examples described above, or it can be realized as a separate component which is then called a *reasoner*. Encapsulating the reasoner as a separate component provides benefits regarding its extensibility and reusability. Chapter 5 discusses several approaches with separated reasoning.

4.3.3. Additional classifications

Three-dimensional model

Schmidt et al. [SAT+99] divide context into three equally important dimensions. The result is a three-dimensional space using *Environments*, *Self* and *Activity* as axes (cf. Figure 4.7). In contrast to the previously described classifications, Schmidt et al. do not differentiate between abstraction levels, but define context as any “knowledge about the user’s and IT device’s state” [SAT+99].

Source-based

Another possibility to classify context data is by source. In their research on imperfect context information [HI04] they provide four possible sources of context information: sensors, human users and derivation from other types. Consistently, they divide context information into categories:

- *Sensed* context is acquired from sensors and usually frequently changing,
- *Static* context is permanently available and never changes,
- *Profiled* context is acquired directly from the users (e.g. in form of profiles) or from his application (e.g. from a history of last user actions),
- *Derived* context is concluded by combining information of the other three types.

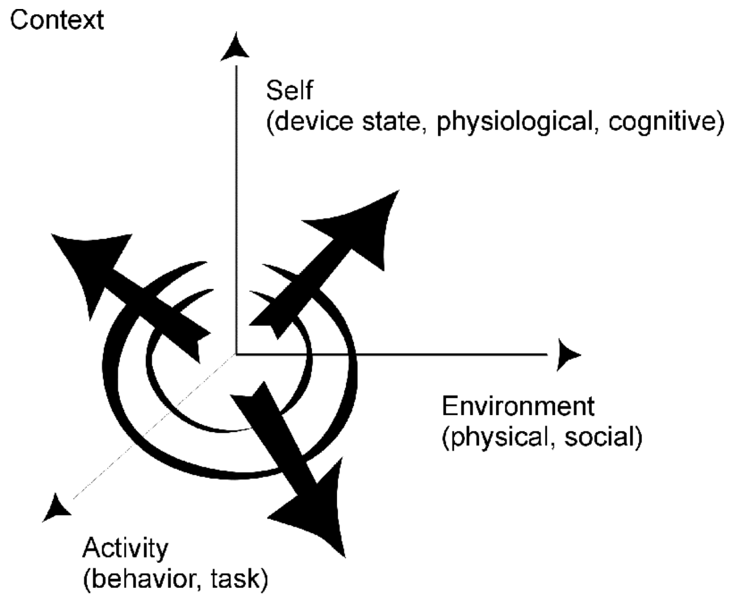


Figure 4.7.: Three-dimensional context classification

4.4. Summary

Context-awareness is a key feature of applications in ambient intelligent production environments. Stemming from natural language, context describes the entirety of circumstantial data that influences the human-computer-interaction. The examples presented demonstrate the usefulness of context in today's working environments, for memory assistance and for ambient information systems. While these applications make use of context inherently in their application logic, they consider only facets of context. Extending their abilities to include more context data always requires additional effort and changing the source code. The reusability is therefore limited. The classification schemes showed that some context types are more important than others. Especially the location of users, objects and devices is of primary interest. The classifications also showed the necessity to combine and abstract low-level sensor data to more meaningful contexts.

The following chapter introduces different approaches to context modeling and reasoning which aim at providing more reusable and extensible software architectures. The separation of context modeling from the application logic follows the *separation of concerns* principle and makes context-aware applications easier to understand and to maintain [Dij20].

5. Context Models and Frameworks

The examples described in Section 4.2 use context data as an integral part of their internal program design and application logic. This integrated approach results in additional development cost for new applications because each one has to map the considered environmental context data to suitable internal structures. Managing the context data has to be done by each application on its own. This is sufficient for small and non-networked devices which use context only in a limited way and which are not likely to change their scope of considered context information over time. But in highly interconnected environments this design leads to incompatibility in the communication between applications. The lack of standards for the exchange of internally represented context information hinders the interaction between such devices.
In this section different modeling approaches and frameworks for the management of context are discussed. The general idea is to encapsulate context information in a separate, application-independent model. Decoupling the context management from the applications ensures the model's extensibility, facilitates the use of a reasoner (cf. Section 4.3.2) and enables its use in MBUID processes (cf. Section 2.3).

5.1. Modeling Approaches

For applications to make full use of context information, they have to be represented in a model. On the formal basis of a context model standardized access and communication is possible. The context model therefore specifies the data structures and extension possibilities for applications. This has substantial influence on the handling of context data. Chen and Kotz [CK00] identified six modeling approaches that were later defined more precisely in a survey by Strang et al. [SLP04]. They differ in their structure and presentation of the modeled data. Each approach has its own advantages and disadvantages which make them suitable for different situations. The following section is based on this classification and gives an overview over the six approaches and example members of the respective class.

5.1.1. Key-Value

Key-value pairs are the simplest approach to modeling. In this approach, each piece of information is assigned to a key value. While the information may change, the key always stays the same. for example to store the current user position a sensor with access to the model would instruct it to store the three Cartesian coordinate values (3.0,2.0,5.0) with the keys ("user_pos_x", "user_pos_y", "user_pos_z"). Another client can then retrieve the coordinates by querying the model with only the keys. This method is very easy to implement. Its simple nature is also of advantage when handling large data sets. The Berkeley-DB [OBS99] makes use of the good reproduction and workload properties of key-value-pairs in database management systems. Already Schilit et al. [SAW94] used this approach to provide context information. While the simplicity makes the implementation very easy, there are also disadvantages that make this approach less attractive for context modeling. First, the saved data lacks structure. While it is possible to introduce a type system for values (e.g. the value associated with the key "user_pos_x" has to be of type float), advanced concepts like references between values need additional effort and introduce new complexity into the system. This approach also requires accessors to have an implicit shared knowledge. To retrieve the user position, the client has to *know* that it is saved in the keys "user_pos_*". The model itself provides no means to discover the correct keys. This means that developers of context-aware applications using the key-value modeling approach need an implicit shared knowledge of the keys used.

5.1.2. Markup Scheme

Markup scheme models are composed of markup-tags annotated with attributes. The contents of markup tags may contain other tags recursively which results in an hierarchical tree-structure. Typical representatives of this category are *profiles*. Usually a serialization of a variant of the *Standard Generic Markup Language* (SGML) [Wor07], the superclass of all markup languages (such as XML), is used to structure profile information in a standardized way. Some of these profiles are defined as extensions to the *Composite Capabilities / Preferences Profile* (CC/PP) [Wor08a] and the *User Agent Profile* (UAProf) [Ope08], which in turn are vocabulary extensions of the *Resource Description Framework* (RDF) [Wor08c]. CC/PP and UAProf aim at describing the *delivery context* of mobile devices. The idea is to tailor delivered contents according to the clients' capabilities, which are described in a profile and provided by a client when it requests content items from the server. So the operator of a web site may for example decide to deliver a simple plain text menu instead of a graphics-heavy implementation if the client profile indicates low screen resolution and CPU power. Listing 5.1 shows an example CC/PP profile for a mobile client with limited screen resolution.

CC/PP describes a basic vocabulary with a range of example resources. Application developers and device producers are encouraged to expand the vocabulary. One CC/PP and UAProf extension is the *CC/PP Context Extension* by Indulska et al. [IRRH03]. They include additional context-relevant information like location, network characteristics, application requirements and session information. While they conclude that their approach is capable of being used in context-aware infrastructures, they also understand that complex context relations are difficult to capture and non-intuitive due to the underlying CC/PP.
The Stick-e-Notes described earlier (see Section 4.2.2) use an SGML derivative for the storage of context-dependent descriptions. This format later evolved into ConteXtML [Rya08]. It defines a protocol for exchanging field notes and other data between mobile clients and a central server. Clients communicate with the server by using ConteXtML documents over HTTP connections. Such documents may include both the client's *current* context and a *required* context. In the current context, clients provide information about aspects of their current environment including location, velocity and details of people involved in the mobile task. The required context represents the client's "context of interest" and effectively defines a filter on the server's database. A client could for example require all field notes regarding a specific zone of interest. ConteXtML also supports basic boolean expressions built into the scheme. Listing 5.2 show a

```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:ccpp="http://www.w3.org/2006/09/20-ccpp-schema#"
         xmlns:ex="http://www.example.com/schema#">

  <rdf:Description rdf:about="http://www.example.com/profile#MyProfile">
    <rdf:type rdf:resource="http://www.w3.org/2006/09/20-ccpp-schema#Client-
         profile" />

    <ccpp:component>
      <rdf:Description rdf:about="http://www.example.com/profile#
           TerminalHardware">
        <rdf:type rdf:resource="http://www.example.com/schema#HardwarePlatform"
             />
        <ex:displayWidth>320</ex:displayWidth>
        <ex:displayHeight>200</ex:displayHeight>
      </rdf:Description>
    </ccpp:component>

  </rdf:Description>
</rdf:RDF>
```

Listing 5.1: CC/PP profile example

ConteXtML query. The `<spatial>` and `<person>` tags at the top provide context information of the client. After that, the `<require>`-part requests all notes recorded by persons with the first name “Andy”.

```
<context session="new">
  <person role="user" first="Kai" last="Bizik" />
  <spatial proj="UTM" zone="32-412" datum="Euro 1950 (mean)">
    <point x="245" y="635" z="12" />
  </spatial>
  <require>
    <note>
      <person role="recorder" first"Andy" />
    </note>
  </require>
</context>
```

Listing 5.2: ConteXtML example [Rya08]

Another approach is the *Centaurus Capability Markup Language* (CCML) [KVH$^+$01]. Kagal et al. created an infrastructure and communication protocol for wireless services. Mobile clients can connect to the nearest Centaurus System (CS) via short-range communication. The CS itself is part of a larger communication network with many other systems. Each CS holds a list of registered services, that is forwarded to the mobile clients and continually updated. Clients can then select an available service, enter the necessary parameters and let the CS execute the service on behalf of the client. CCML is used as a service interface description language in the communication between CS and client as well as in the communication between the CS nodes. While CCML is mainly used as a communication interface, the authors already envisioned a more intelligent brokering of services that takes the location of the mobile device into account and orders the service list accordingly.

The biggest advantage of markup schemes lies in their versatility and availability on many platforms. The ability to universally structure information is a very strong argument for using them in distributed environments. As their structure can be described in scheme definitions, automatic validation of the whole or of parts of the model is easy. Managing markup models is facilitated by a number of available software tools. For the popular XML different query languages like XPath [Wor08d] and XQuery [Wor08e] as well as transformation languages like XSLT [Wor08b] are available that can be used to easily query or transform the contained data. One drawback is that unambiguity and and incompleteness of information have still to be handled proprietary by the applications.

5.1.3. Graphical

A very well known tool for modeling is the *Unified Modeling Language* (UML) [Obj08]. UML diagrams have a strong graphical orientation and are often used in software development processes. Due to their universal structure, they can be used in a wide range of domains. They have also been used to model context. Bauer [Bau03] uses UML extensions to model context specific aspects in air traffic control (ATC). Figure 5.1 shows his UML diagram modeling the part of his context that includes flight trajectories and possible conflicts between aircrafts.

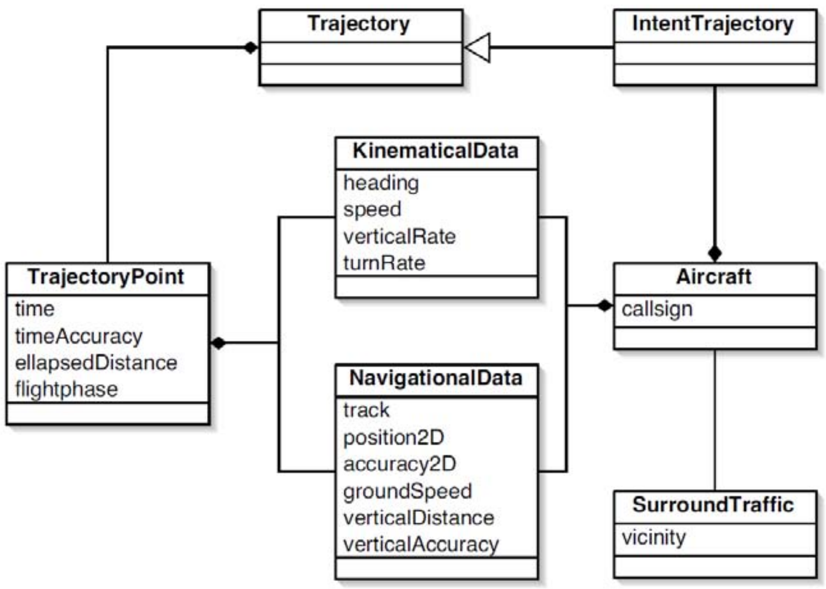
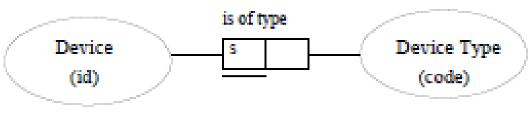
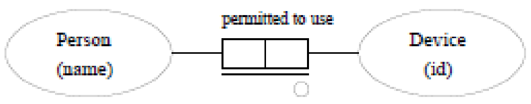


Figure 5.1.: UML model in air traffic control [Bau03]

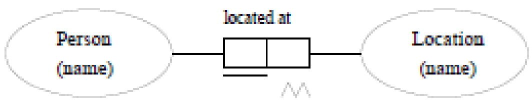
Another graphics oriented context model is the extension of *Object Role Modeling* (ORM) by Henricksen et al. [HIR03]. ORM [ORM08] is originally a fact-oriented tool for system analysis at conceptual level with a strong focus on intuitive handling and on using natural language for easy understandability. Henricksen et al. extended ORM to capture context-specific needs like histories or uncertainties of information. Figure 5.2 shows the full capabilities of their extension.



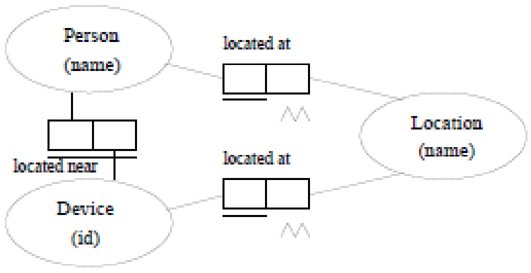
(a) Static Fact Type



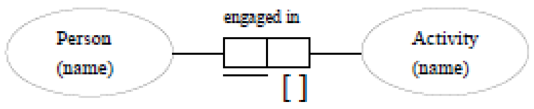
(b) Profiled Fact Type



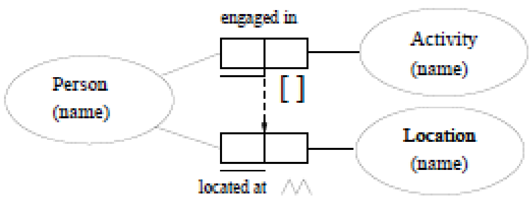
(c) Sensed Fact Type



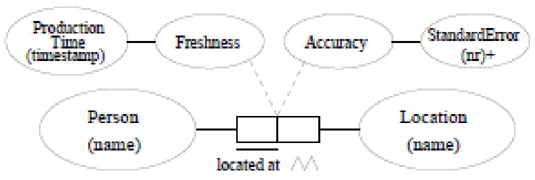
(d) Derived Fact Type



(e) Temporal Fact Type



(f) Fact Dependency



(g) Quality Annotation

Figure 5.2.: Extended ORM with new fact types [HIR03]

The basic modeling concept in ORM is the fact. *Entity types* are shown as ellipses, simple values that do not need a reference scheme as dashed ellipses. *Fact types* are shown as sequences of role boxes, where each box is connected to an entity type. The extension allows fact types to be categorized as *static* or *dynamic*. Static facts remain unchanged as long as their entity types exist (e.g. "Device *id* is of type *d*"). Dynamic facts are further categorized by their source as *profiled*, *sensed* or *derived*. The other extensions include the capturing of temporal facts ("Person *x* was engaged in activity *y* for three hours"), dependencies on facts ("If person *x* engages in another activity, their location is likely to change.") and quality annotations (how fresh or accurate a fact is). This type of modeling is especially useful for deriving an *Entity Relationship* (ER) model from it, which can be used as a basis for a relational database layout. Henricksen et al. provide methods to map their extensions to ER-models. However, they do not provide any software tools to manage context models with extended ORM-diagrams. This is also a general problem with other graphical modeling methods. While there are sophisticated software tools for well known languages like UML and ORM, the use of extensions beyond the standard capabilities implies additional adjustments in the modeling software. Graphics-oriented models also need to be transformed for automatic use. Devices need a serialized version of the diagrams to communicate. For this purpose, instances of graphical models are usually serialized in more machine-readable formats like XML. However, if the necessary tools are provided, graphical diagrams are a very intuitive and natural way of modeling. Graphical approaches are used mainly for human structuring purposes.

5.1.4. Object-oriented

Object-oriented models aim to retain the basic paradigms of object orientation: Encapsulation, reusability and inheritance. Structurally related context information is encapsulated in an object's internal state and can only be accessed by its public interface. Therefore the exact details of data processing are hidden from the clients. This is ideal for refining low-level data from sensors to a useful degree of abstraction. Schmidt et al. [SAT+99, SBG99] use an object-oriented approach in the TEA-project's[1] layered context architecture (Fig. 5.3).

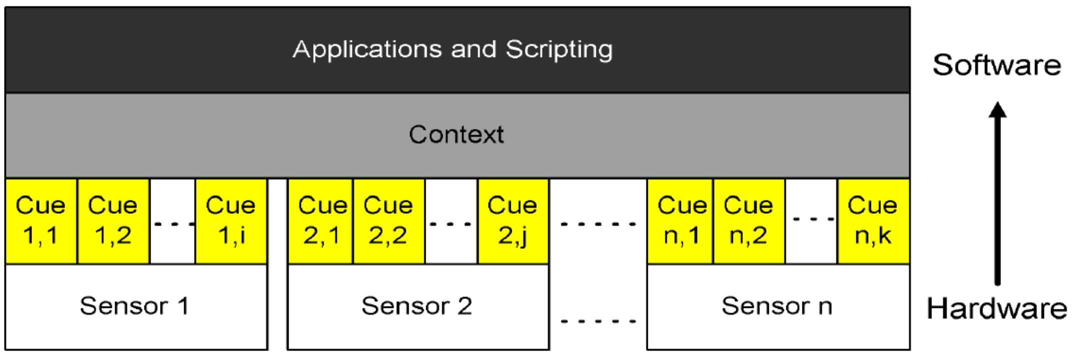


Figure 5.3.: TEA context architecture [SAT+99]

[1]Technology for Enabling Awareness [Esp08]

So-called *Cues* are the linking layer between raw sensor data and the context recognition layer. They are regarded as time-dependent functions that collect sensor data from physical and logical sensors. Then they perform statistical analyses before the next layer concludes the definite context situations from their output. Each cue collects data from exactly one sensor, but sensors may provide their data to multiple cues. Also, each cue has a defined symbolic or sub-symbolic output with an infinite or finite set of possible values acting as the Cue's interface. The "Applications and Scripting" layer at the top provides mechanisms for applications to include context information. Three scripting primitives have been implemented that can be used to trigger application-specific actions upon entering, leaving or while in a context[2].
Object oriented context models are well suited for the use in distributed environments. The model can easily be extended by new information types (classes) or updated information (object) and the interfaces ensure a sufficient level of formality. Validation is usually done by a compiler (structural level) and a runtime environment (instance level). The drawback is that all participating clients need to use the same programming language or have to serialize the information for exchange.

5.1.5. Logic-based

Logic-based modeling is the most formal approach to context modeling. It usually consists of basic facts (axioms), expressions and rules that define on which conditions (combination of basic expressions) a concluding fact may be derived. The process of concluding by defined sets of rules is also known as *inferencing* or *reasoning*. Changes in the environment are reflected in the model by adding, removing or updating the facts and respective rules.
One example of a context model based on logic is given by Bacon et al. [BBH97]. Their model is based on an ER-model of the environment and aims at supporting a variety of multimedia applications such as forwarding an active user session to the nearest workstation. Clients can register with the model by providing *event templates*. The parameters of these templates may contain wildcards. The clients are notified if there is a change in the specified range of interest specified by the template. Their system is implemented in Prolog and also supports direct logical queries. Listing 5.3 shows an example of a logic based query. The lines one to five declare basic facts (such as "John Bates is a person"), line 6 defines a rule for determining if a person has access to a video-capable workstation in the same room. The last line is the query that - in this case - would evaluate to the boolean value true. Logic based models are well suited for applications in the domain of artificial intelligence. But they are also the least intuitive way of modeling and resource intensive - especially in models with a high number of rules.

5.1.6. Ontologies

Ontologies represent knowledge by formally defining the relations between terms. The term "ontology" itself stems from philosophy in which it describes a theory about the nature of exis-

[2]A single context in their definition is basically an abstract description of a use situation. Their system models context as a two-dimensional vector that contains many contexts and a probability value indicating how likely the respective situation currently applies.

```
person(john.bates)
workstation(britten)
has_video(britten)
room(t15)
location(john.bates,t15)
video_in_room(P) :- location(P,L), location(W,L), workstation(W), has_video(W)
video_in_room(john.bates)
```

Listing 5.3: Example query of a logic based context model in Prolog [BBH97]

tence. “Ontology refers to the discipline that deals with existence and the things that exist. In computer science things that ‘exist’ are those which can be represented by data” [Ay07]. The typical ontology used in computer science consists of a taxonomy and a set of inference rules. Similar to inheritance concepts in object oriented programming, the taxonomy defines classes and relations between their members. For example a “student” may be one type of “person” and be related to a “research group”. Inference rules add additional power to ontologies. Supported by the idea of the *Semantic Web* [LHL01], different languages for describing ontologies like *RDF Schema*(RDF-S), the *DARPA Agent Markup Language* (DAML) and the *Ontology Interchange Language* (OIL), which were later joined and then superseded by the *Web Ontology Language* (OWL) emerged.
An early example of research using ontologies to model context information is the work of Öztürk and Aamodt [OA97]. In their psychological studies they explored the difference between recall and recognition regarding several circumstances in combination with contextual information. They were confronted with the necessity to merge knowledge from different domains and chose ontologies because of their formality as solution approach. They describe the application of their approach in clinical case-based reasoning and problem solving.
The *Aspect Scale Context* (ASC) is another ontological context-model. Strang et al. [SLPF03] use it as a core part in their “Context Ontology Language” (CoOL). ASC provides support for specifying the model’s core and an arbitrary number of sub-concepts and facts. Different *Scales* of one *Aspect* must be connected by an “IntraOperation”. So the specification of equivalences between scales is possible[3]. Figure 5.4 shows ASC’s core classes and relations. The modular design was chosen with the aim to enable context sharing and reuse in ubiquitous computing systems.

The *Context Ontology* (CONON) approach by Wang et al. [WZGP04] is based on the same idea. The focus here is also on knowledge reusing and sharing. CONON uses OWL as description language and consists of an *Upper Ontology* containing location, user, activity and computational entity as core classes and a *Lower* or *Domain-Specific Ontology.* The specific ontology is a collection of ontologies defining complementing details and features of each subdomain.
A very comprehensive ontology is the *Standard Ontology for Ubiquitous and Pervasive Applications* (SOUPA) by Chen et al. [CPFJ04]. It heavily relies on OWL’s import capabilities to extend the SOUPA Core. With the help of modular component vocabularies it intends to

[3]For example the Aspect “Temperature” may have the scales “Celsius” and “Kelvin” with an IntraOperation to convert between them.

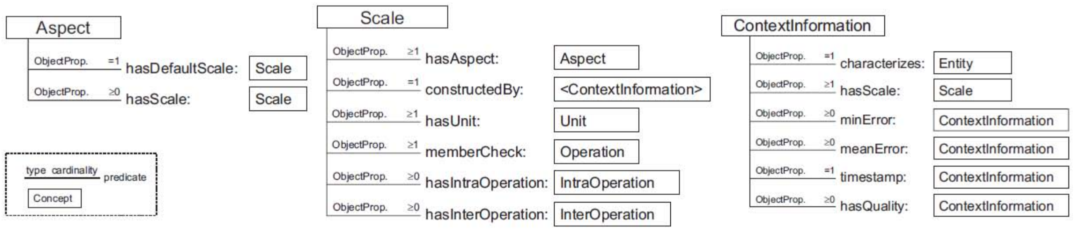

Aspect
ObjectProp. =1 hasDefaultScale: Scale
ObjectProp. ≥0 hasScale: Scale
type cardinality predicate
Concept
Scale
ObjectProp. ≥1 hasAspect: Aspect
ObjectProp. =1 constructedBy: <ContextInformation>
ObjectProp. ≥1 hasUnit: Unit
ObjectProp. ≥1 memberCheck: Operation
ObjectProp. ≥0 hasIntraOperation: IntraOperation
ObjectProp. ≥0 hasInterOperation: InterOperation
ContextInformation
ObjectProp. =1 characterizes: Entity
ObjectProp. ≥1 hasScale: Scale
ObjectProp. ≥0 minError: ContextInformation
ObjectProp. ≥0 meanError: ContextInformation
ObjectProp. =1 timestamp: ContextInformation
ObjectProp. ≥0 hasQuality: ContextInformation

Figure 5.4.: Aspect-Scale-Context Model [SLPF03]

represent intelligent agents with associated beliefs, desires and intentions, time, space, events and user profiles. The often neglected inclusion of policies for security and privacy issues is also considered. The authors demonstrate one of the greatest advantages of the ontological approach by extending the SOUPA Core with a number of ontologies that were developed to support other pervasive computing scenarios. The standard OWL mapping constructs (e.g. `owl:equivalentClass` and `owl:equivalentProperty`) are used to map to the vocabulary extensions. Thus, the interoperability between SOUPA applications and applications based on other ontologies is ensured. An overview over SOUPA is shown in Figure 5.5. The ontologies already imported by Chen et al. include the *Friend-Of-A-Friend* ontology, *DAML-Time*, the spatial ontologies from *OpenCyc*, *Regional Connection Calculus* (RCC), COBRA-ONT, MoGATU BDI and the Rei policy ontology[4].

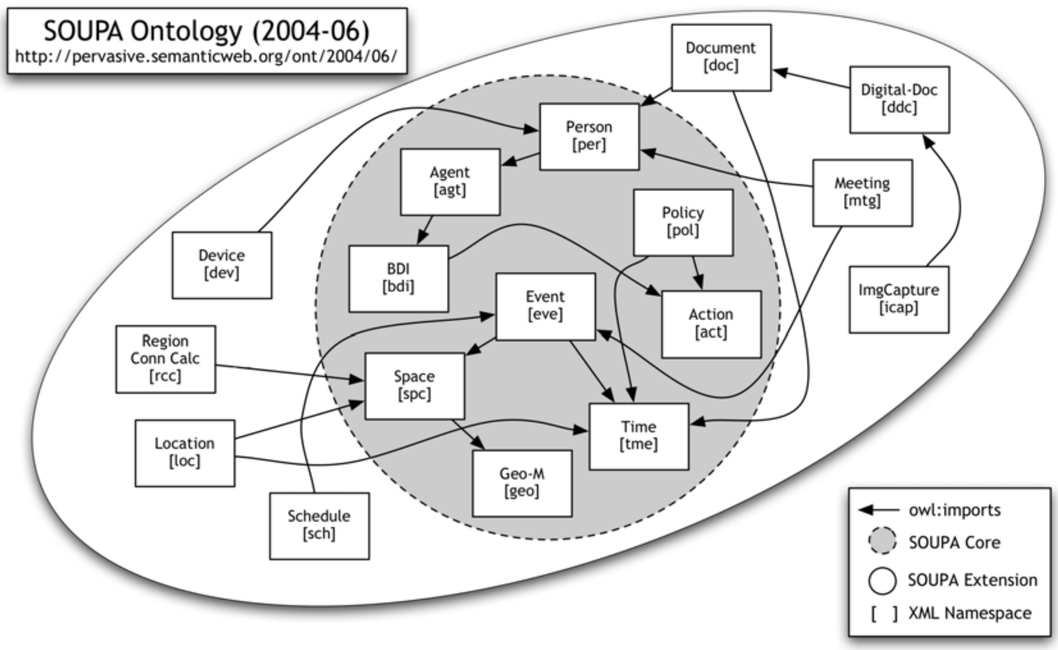


Figure 5.5.: SOUPA Ontology with SOUPA Core and SOUPA Extension

[4]The full list with references can be acquired from [CPFJ04]

With the rising interest in ontological modeling and the availability of OWL as description language, a number of tools have been developed to support the work with ontologies. Protégé [Sta08] is a graphical editor, Jena [Jen08] provides an API to access OWL ontologies in Java and Racer [HMHW08] is an inference engine that can be integrated into Protégé and Jena. Ontologies are a very powerful modeling tool that support a wide range of applications. However, their power comes at the cost of poor scalability. While the specification and serialization of ontologies is relatively easy in languages like OWL, the inferencing is based on logic and mostly realized as external reasoning component. The authors of the ASC-model even require a conversion from CoOL in F-Logic [KLW95]. Their argument, that this is not a real disadvantage of their system because comparable approaches have similar requirements [SLPF03], hints at a general drawback. The observations made by Wang et al. with CONON also confirm the low performance. They conducted empirical tests with different computers and variously sized ontologies. The response time of their system ranged up to about eight seconds. Their recommendation is to use ontologies only in non-time-critical scenarios [WZGP04]. In addition, vocabulary extensions are not always completely compatible with existing ontologies. While integrating other ontologies into the SOUPA Extension, Chen et al. described that they had to make structural adaptations, because a direct mapping was not sufficient for some concepts [CPFJ04].

5.2. Location models

The inclusion of location data was one of the first aspects of context awareness (cf. Section 4.1.2). In smart production environments with self-organizing modules awareness of components' positions in the room and in relation to other objects is of vital importance. Becker and Dürr introduce a classification of context models in ubiquitous computing [BD05]. They describe different properties of coordinates. In general, *geometric* and *symbolic* coordinates can be distinguished.

- *Geometric coordinates* are given as coordinate tuples relative to a reference system. The World Geodetic System 1984 (WGS84) is an example for a global reference system. It can be used to specify locations anywhere on this planet. Cartesian coordinates, in contrast, are often used locally (e.g. within one room), because the calculations for distance and the relative position of objects is less complex. Geometric coordinates allow spatial reasoning, such as the distance to the next-nearest object or the overlapping of areas.

- *Symbolic coordinates* define positions with the help of symbolic identifiers. Room numbers and street names are symbolic coordinates. Spatial relationships between two symbolic positions are not defined by default. Only with additional knowledge topological information can be derived. For example, if the areas described by symbolic coordinates are pairwise disjoint, it can be concluded that two objects with different coordinates are not adjacent to each other.

Following the authors' argumentation, a location model has to support queries for (a) a specific position, (b) the n nearest objects to a given position (nearest neighbor query) and (c) interconnections between locations. Based on these queries, Becker and Dürr describe, what a location model should provide:

- *Object positions* in the form of geometric and symbolic coordinates as well as one or multiple local and global reference systems.
- A *distance function* to determine the distances between modeled objects.
- *Topological relations*, namely spatial containment (for range queries) and spatial connections (for navigation).
- Optionally, the *orientation* of objects.

The final taxonomy contains four types of symbolic location models and two hybrid (symbolic/geometric) types. The following enumeration mostly follows the authors' argumentation in [BD05].

Set-based

Set-based location models are based on a set C of symbolic coordinates. This set describes the entirety of known positions. Locations spanning multiple coordinates are modeled as subsets of C. Figure 5.6 shows a floor plan as an example.

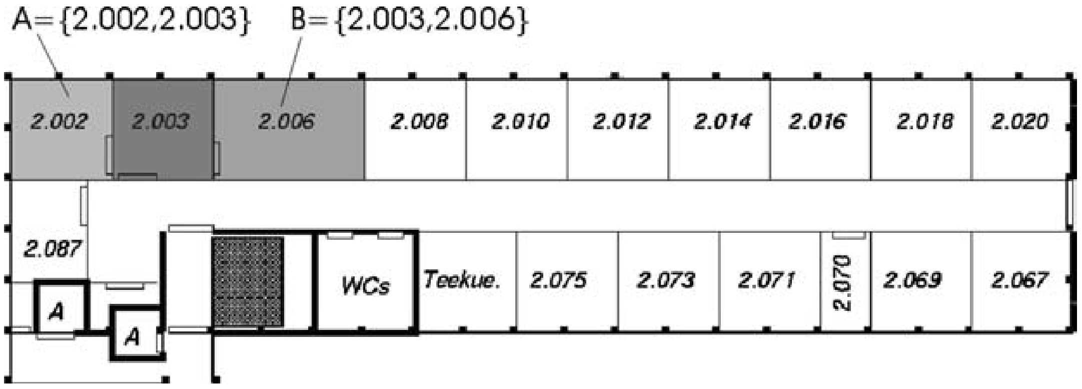


Figure 5.6.: Example for a set-based location model: Floor plan with symbolic coordinates [BD05]

In such a model containment relations can be determined by calculating the intersection of two sets, but only a rudimentary qualitative expression concerning the distance between two coordinates can be made. Statements like "a and b are twice as far from each other than c and d" cannot be evaluated. Set-based location models have the least expressive power.

Hierarchical

In hierarchical models the set of known coordinates is ordered according to the spatial containment relation. The topmost location is therefore the symbolic coordinate "everywhere". Traversing the set means excluding locations. If the areas are pairwise disjoint, the hierarchical model depicts a tree-based structure. Extending the previous example, a building could be modeled by containing several floors with different rooms. As this kind of model is based on spatial inclusion, range queries are naturally supported. In hierarchical models, interconnections between locations cannot be modeled and therefore distance has still only qualitative character.

Graph-based

Graph-based models are based on a directed graph $G = (V, E)$, where the coordinates define the graph's vertices V. Edges between vertices are added if a direct topological connection exists between the corresponding locations. Therefore the modeling of the "connected to" relation can be regarded as intrinsic. The distance between two locations a and b can be measured as the length l of the shortest path $P = (a, p_1, p_2 \ldots p_{l-1}, b)$ from a to b. Additionally, the edges may be weighted to model transport costs or distances, which allows graphical algorithms to find shortest paths according to the weights used. Nearest neighbors are also easy to determine. Range queries are possible, but the range itself needs to be defined. For example an upper bound for the distance can function as criterion. The resulting query would return all locations whose distance from the base location does not exceed the limit. What graph-based models lack is the possibility to explicitly model containment relations.

Combined

The advantages of set- and graph-based location models can be combined. In the set-based part of a combined model locations are modeled as subsets of the superset containing all symbolic coordinates. On this part, range queries can easily be executed as described above. The graph-based part overlays the sets. Here, locations are connected if a physical connection in the real world exists (like a door between rooms, stairs between floors, etc.). Figure 5.7 shows the previous floor plan example enhanced with a directed graph to model access paths. Nearest neighbors, distances and even navigational applications are executed on the graphical part of the model.

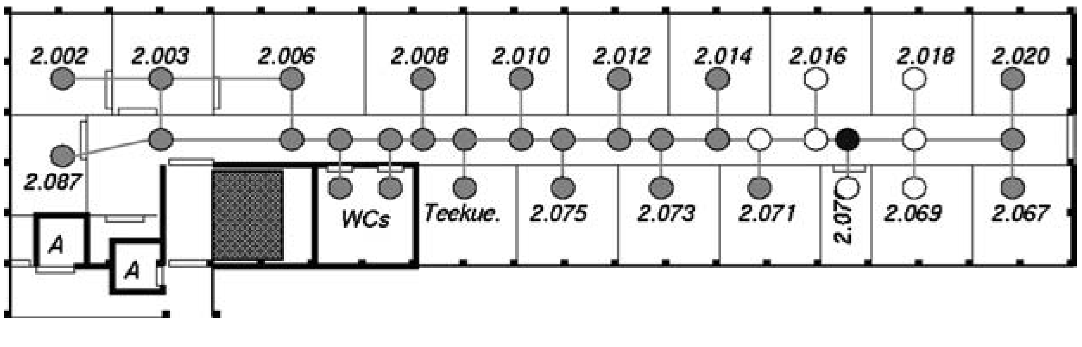


Figure 5.7.: Example combined location model: Floor plan with access paths as directed graph [BD05]

Subspaces

This is the first hybrid location model. The basis is still a symbolic location model as described above. But a hybrid model also contains the geometric extent of each location. This extent can be noted with respect to a global coordinate system (like WGS84) or locally with limited scope. Subspaces are then created by anchoring coordinate systems in other coordinate systems. A building could be modeled by giving an anchor point for each floor (like the position of a staircase) and then using relative coordinates for the rooms on each floor. This provides a better clustering of the model as coordinates in the same local system can be compared easily while the translation from local into global coordinates remains possible. Therefore no sacrifices concerning the expressiveness are made. On the contrary, models that are structured this way are usually easier to read by humans.

Partial subspaces

Partial subspaces are equivalent to normal subspaces with the exception that not all locations need to have associated geometric coordinates. This is a very flexible approach, as the level of detail can be easily varied. For example, a coordinate system spanning the whole plant grounds could be used as the topmost reference system. When searching a person in a small building, it might then be sufficient to know the building's entrance coordinates and the room number. In this case only the entrance's geometric coordinates have to be known.

5.3. Frameworks

The encapsulation of context information in a separate, formal model is the first step towards better reusability. While the modeling approach technology has a substantial influence on the formal representation and therefore the *storing* of context data, there is also the aspect of *managing* it. Managing is the process of acquiring context data from sensors, saving it in the context model, internally transforming the model to ensure the up-to-dateness of basic and derived data, distributing it between connected members of the context-aware system and providing convenient means for accessing it. Ideally, a platform-independent infrastructure handles these details transparently from the applications. Such an infrastructure stands "in the middle" between sensors and the applications built on top of it. It embodies an abstraction layer and is therefore called *middleware*. This layer may be composed of multiple interacting systems as shown in Figure 5.8, but it does not need to be so. It is interesting to note that applications making use of the middleware do not necessarily need to know all systems. They can rely on discovery mechanisms or on locally available data.

The following frameworks explore some of the described aspects in greater detail. They were proposed to facilitate the creation of context middleware and context-aware applications. This selection was chosen because of the variety in the approaches. The *Context Toolkit* tries to transfer the widely used concept of *widgets* from the domain of graphical user interfaces (GUI) into context modeling. *Hydrogen* addresses some problems with the toolkit's approach that result from the mobile nature of many context-aware devices. The *Java Context Awareness*

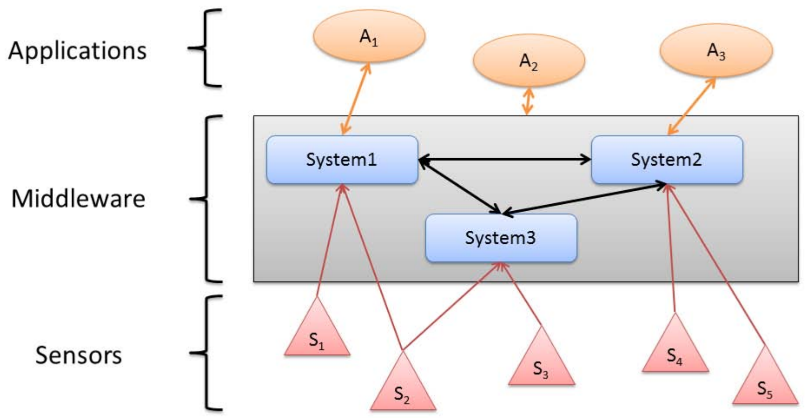


Figure 5.8.: Basic principle of context middleware

Framework is an event-based system that abandons parts of the platform-independence but can be extended during runtime.

5.3.1. Context Toolkit

Salber et al. presented the object-oriented *Context Toolkit* that was developed at the Georgia Institute of Computing [SDA99]. They transferred the concept of *widgets* from the domain of graphical user interface programming to context modeling. GUI widgets are available for a large number of frequently needed solutions: Selecting a file, triggering an action or choosing from a range of options. The authors try to convey the main benefits - transparency, reusability and abstraction - of GUI widgets to context modeling. The same way GUI widgets separate applications from the details of the underlying system (cf. Figure 5.9), *context widgets* separate applications from context acquisition details.

Context widgets have a *state* and a *behavior*. One example could be an `RFIDTagData` widget in the *SmartFactory*^{KL}. Its state would consist of variables for its location, the last read data string of passing RFID tags and the last time an RFID tag was detected. The standard behavior would be to read all data from an tag as soon as it comes into range, to update the variables accordingly and then to notify registered listeners of the change. The widget approach is very flexible as new widgets can be added without interfering with the rest of the system. The distributed nature of ambient intelligent environments was also taken into account by allowing the widgets themselves to be distributed. One single widget may be composed of a number of sub-components of three types: *Generators*, *Interpreters* and *Servers*. So, while the application programmers perceive a single widget, in reality the widget's functionality

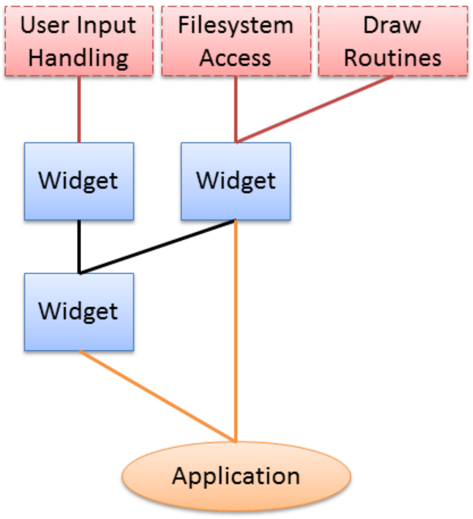


Figure 5.9.: Abstraction concept of GUI widgets

may be distributed over different systems. The `RFIDTagData` widget would typically contain a *Generator* component that handles the hardware details and connection to the RFID reading device. Similar to the *Cue* concept of the TEA-project (see Section 5.1.4) *Interpreters* are used to convert low-level sensor data to a semantically meaningful representation. For example a `FillLevel` widget could contain a generator component, that supplies the weight of passing soap bottles from a scale integrated into the conveyor, and use an interpreter to convert these values into one of the three states "no bottle present", "bottle empty" and "bottle full". Finally, *Server* components act as an aggregator. They collect, store and interpret other widgets' information. In the example above a `BottleState` widget could combine the results of the other two and only clear an application to start filling the currently passing bottle if it is empty and the data on its RFID-tag holds an according instruction. The whole example is shown in Figure 5.10.

The authors demonstrate the toolkit's feasibility by implementing two prototypes and remodeling one existing application. Some advanced features like preserving historical context data and supporting a heterogeneous environment with a standard communication mechanism (e.g. "XML over HTTP") are already considered in the design. But there are also weaknesses. The perhaps most important ones are the lack of a discovery mechanism for new resources and the problems arising from unreliable networks. If an application loses the connection to a widget or widget components lose contact with each other, the subscriptions to status notifications have to be restored manually. This is not unlikely to happen with mobile clients, especially in areas with lots of wireless networks and high interference levels. Applications relying on remotely available widgets cease to function on connection loss.

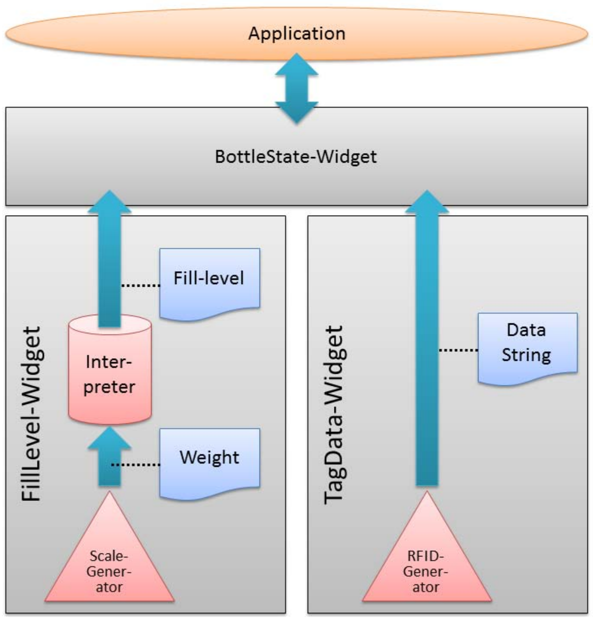


Figure 5.10.: Example of an context architecture built with the context toolkit

5.3.2. Hydrogen

Some weaknesses of the Context Toolkit are addressed by Hydrogen [HWM+03]. Hydrogen is designed for the exclusive use on mobile devices. All three layers of an instance of the framework run completely on one mobile device. The author's focus lies on taking the resource restrictions and the ad-hoc nature of communication in ubiquitous systems into account. Hydrogen is therefore a very *lightweight* (low resource consumption) and *robust* (against network disconnections) framework. A mechanism for *context sharing* with other devices running Hydrogen has also been proposed. The framework's architecture is shown in Figure 5.11.

The *Adaptor Layer* fetches sensor information, possibly performing first evaluations of the raw data, and then forwarding it to the next layer. Each sensor is queried by only one Adaptor, thus avoiding conflicting access. The Adaptors themselves are reusable objects that can be queried by many instances of the framework. These queries and the distribution of sensed data is handled by the *Management Layer*. The *Context Server* is located in this layer. It stores all gathered information about the environment and provides the necessary means for applications to access it. Both, asynchronous queries and synchronous subscription-based retrieval are supported. The *Management Layer* also has the ability to share sensed contexts with other devices in range. Naturally the authors classify context data as *local* or *remote*. The exchange of context data via peer-to-peer communication enables the inclusion of information in the applications' logic that local sensors alone do not provide. Whenever another device is encountered, there can be a mutual exchange of information. Thus, a local representation of the remote device's context is established and can be used in combination with the local model. By default, the framework distinguishes between five types of context: Time, Location, Device,

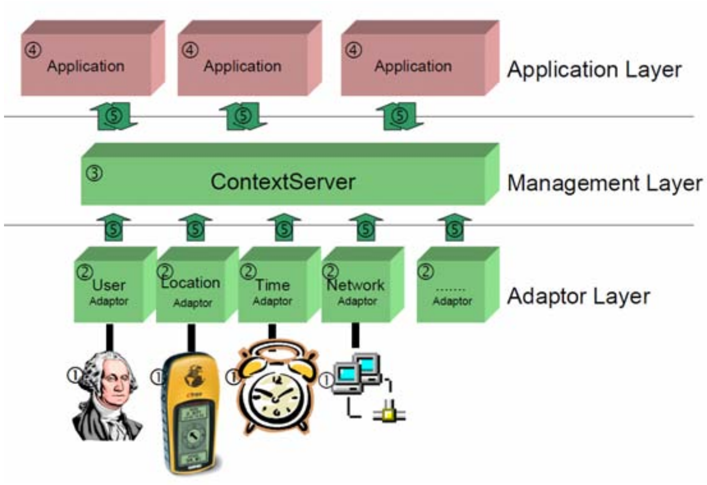


Figure 5.11.: Three-tier architecture of the Hydrogen framework [HWM+03]

User and Network. The framework is written in Java and the context types are represented as Java objects. The class hierarchy is shown in Figure 5.12.

Extending the framework with new types of context is done by adding Java objects which extend the `ContextObject` class and implement two abstract methods, namely `toXML()` and `fromXML()`. XML is used as exchange format on the Management Layer and in the communication between non-Java applications and the Context Server. While Java applications can easily use existing classes for the retrieval of context information from the Context Server, developers need to implement their own routines to interpret the received XML data in other languages. The authors demonstrate the feasibility of their approach with a reference implementation of a context-aware postbox that adapts transmitted data according to screen size and network bandwidth.

The use of a locally running framework provides benefits regarding the availability of context information. Context that has been sensed before is available even if the sensors that originally provided the information have been disconnected. Hydrogen clearly takes the right steps in this direction. However, the sharing of context is still an open issue. The exact details, which instance shares what data with whom and how frameworks communicate their specific range of interest, are not clear. The problem of conflicting data is also only mentioned briefly. In an ad-hoc network infrastructure with occurring disconnects it is likely that two devices that just connected have different local representations of remote context data. This can happen because the remote context was sensed at different points in time or because of inaccuracy (two temperature sensors at the same location will always sense slightly different temperatures). Hofer et al. recognize this problem, but give no solution. They do not consider sensors that are not integrated into the local device either. A number of sensor systems are supplied with support for wireless communication but without the possibility to integrate additional software. The connection of an Adaptor to such a system stays vulnerable to disconnects.

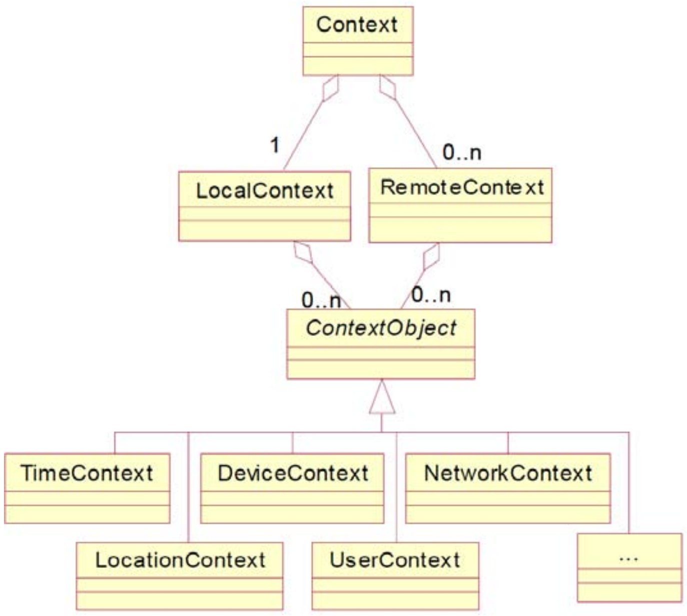


Figure 5.12.: Hydrogen's class hierarchy [HWM+03]

5.3.3. JCAF

The *Java Context Awareness Framework* (JCAF) [Bar05] is a Java-based framework. It forgoes the interoperability with other programming languages and focuses on the management of context information as Java objects. In return, it can make full use of already existing Java concepts like Remote Method Invocation (RMI). Additionally, it can use the serialized form of Java objects without the need to deal with other formats or any markup languages. The goal of JCAF is to provide a lightweight, event-based and secure system with an expressive, but compact set of interfaces.

JCAF is distinctive in three ways: First, it has a loosely-coupled, service-oriented infrastructure. Distributed context services cooperate in a peer-to-peer manner. Second, the framework is extensible *at run-time*. Services, monitors, actuators and clients can be added to a running system. And third, the JCAF-API[5] provides expressive interfaces which developers need to extend the framework with application-specific behavior. JCAF's general runtime architecture is similar to the previous two approaches (cf. Fig. 5.13).

The *Sensor and Actuator* layer is responsible for collecting, pre-processing and forwarding sensor data. The *Service Layer* contains multiple cooperating *Context Services* that derive high-level context information from the sensor data and provide client access via the *Context Service API*. Finally, the *Client Layer* contains the context-aware applications. The right-hand side of Figure 5.13 shows the internal structure of a context service. *Entities* are the main components that abstract context information is associated with. They are contained in the *Entity Container* which controls their lifecycle. *Transformers* and *Aggregators* are small application-specific Java programs developers can add to the *Transformer Repository* at run-

[5]Application Programming Interface

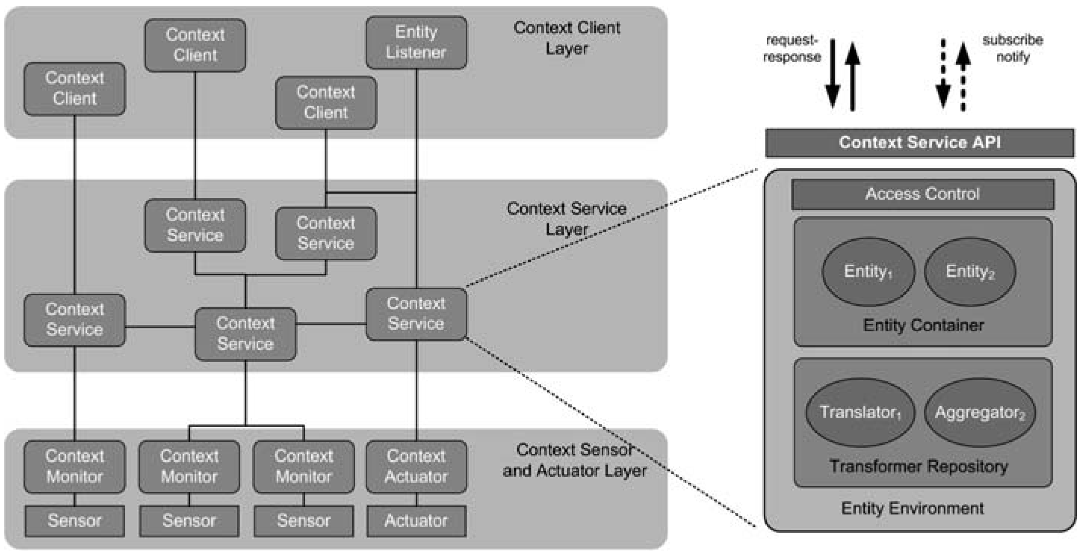


Figure 5.13.: JCAF Runtime Architecture [Bar05]

time. Other clients can then query the repository, for example in case a context information has to be transformed from one format into another. Access to the whole context service is regulated by an *Access Control* component. It implements authentication based on the *Java Authentication and Authorization Service* (JAAS). The access is restricted by a coarse-grained role-based control mechanism, which labels context information as secure if its source is a trusted Context Monitor. The Service API supports three query methods: Clients can access the entities by following a request-response schema, by subscribing as *Entity Listener* to specific entities or by subscribing to all entities of a specific *type*. Entities and their context information as well as items in the Transformer Repository are remotely available. All connections drawn in Figure 5.13 are remote connections realized with Java RMI.

JCAF follows the object-oriented approach to context modeling. Information is modeled by extending generic classes. The whole class diagram is shown in Figure 5.14. The important interfaces are `Entity`, `Context`, `Relation` and `ContextItem`. The abstract classes `GenericEntity`, `GenericContext`, `GenericRelation` and `AbstractContextItem` implement these interfaces and can be specialized by developers to add additional types. An entity is the representation of a physical entity in the real world (e.g. a person). One *Entity* is associated with exactly one *Context*. A single *Context* is composed of *Relations* to multiple *ContextItems*. The lowermost row in Figure 5.14 contains examples for the various types. With this structure, it is possible to model that the person *Kai* (which is an entity) is *Located* (relation) at "Room 32-412" (ContextItem). The `EntityListener` interface contains the `contextChanged()` method which is called if the context of an observed entity changes. JCAF has been evaluated and used in a number of projects [Bar05].

JCAF covers many issues, the Context Toolkit and Hydrogen do not. The exclusion of programming languages other than Java enables the usage of RMI and serialized objects, which greatly facilitates the communication between services. With the integration of authentication and authorization via JAAS, JCAF also provides security between connected services - an

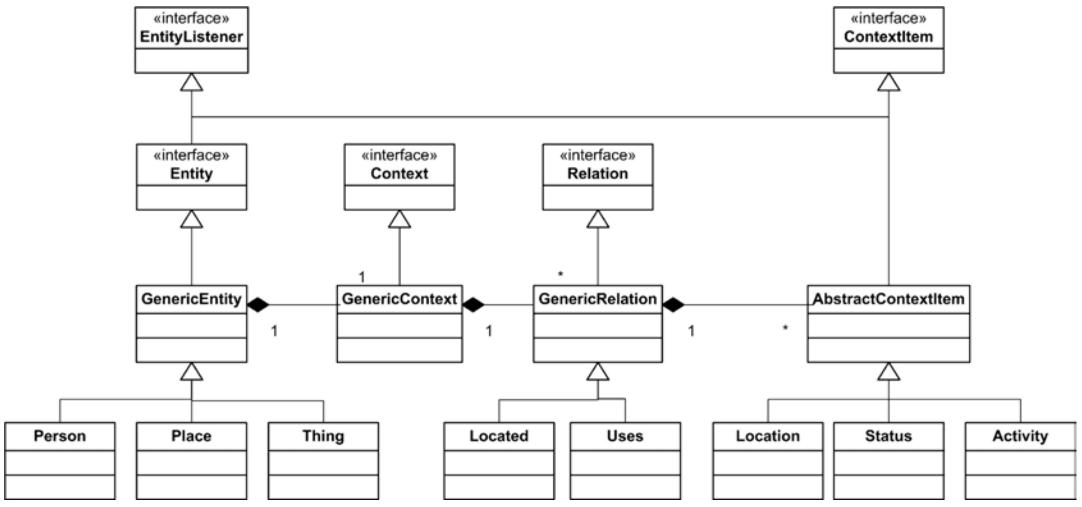


Figure 5.14.: JCAF's context model as UML diagram [Bar05]

issue neglected by most frameworks. The currently simple role-based security mechanism is only a minor drawback as it can be replaced by other mechanisms like Access Control Lists (ACL) with little effort. One real weakness is the absence of a discovery mechanism. The Context Services do cooperate in a peer-to-peer fashion, but a list of available services has to be supplied at startup. A dynamically changing network topology will introduce further concerns if the context data of a single entity is stored across multiple systems. The synchronization of such data is not trivial. The `ContextItem` interface also enforces items to implement the `getAccuracy()` method for quality indication. This is, however only a numerical value between one and zero. This might be insufficient because it does not contain any meta-data of how the value was calculated. Finally, the services provide no framework-wide means to access the history and contexts from previous points in time. In one project, this was solved by adding a designated *History Service* that monitored entities of interest and remembered every change in their context.

5.4. Other Context Frameworks and Middleware

With the ongoing research in context-awareness the number of available context models and frameworks is continually increasing. A detailed discussion of all approaches goes beyond the scope of this thesis. The following section briefly outlines additional interesting approaches to context-aware systems. Especially *ontological* and very comprehensive approaches that require a whole network of servers dedicated to managing, are too resource-intensive for a context engine on a mobile device. For details, the reader is referred to the respective sources and the surveys in [BCQ$^+$07, BDR07].

ACTIVITY In [KO04] the authors introduce a novel concept for modeling context. During their analysis of existing approaches they discovered that the research on context often concentrates on isolated elements like the location and their influence on context-aware applications. The *relation* between these elements and their effect on ambient environments have been neglected. They propose a context model based on *Activity Theory*, which allows the description of key influences on human activities. Consistently, they regard context as the set of influences that affect the users' intentions while performing a task. Context is modeled in a triangular structure based on the original activity theory approach (cf. Fig. 5.15). Unfortunately, no formal definition of the model is given.

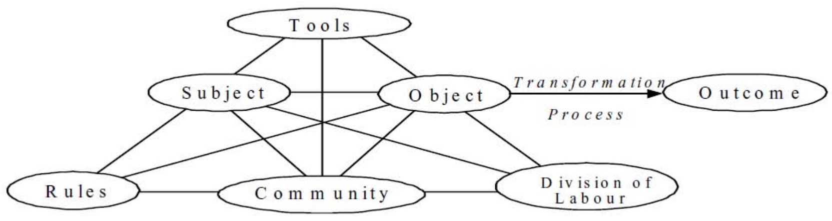


Figure 5.15.: Activity Theory structure as context model

CoDAMoS The *Context-Driven Adaptation of Mobile Devices* project uses an ontological and extremely general context model [PvdBW$^+$04]. The two-layered model is used in the PACE middleware [HIMB05]. The four main entities are: (a) *user* as central concept in many context-aware systems, (b) *environment* as description of the users' surroundings, (c) *platform* as description of the used devices' capabilities, (d) *service* as available functionality in the environment. It is expressed in the *Web Ontology Language* (OWL).

Nexus Many context-aware systems are coarse-grained or use constraints tailored to the applications' requirements to keep the complexity and performance acceptable. In contrast to these approaches, the Nexus Platform [DR03] aims to provide a shared global and detailed context model. Partial models from various providers are federated into a *spatial world model.* The platform consists of three tiers. On the lowest level, the *Service Tier* contains *Context Servers.* They hold partial models with information about their environment. A Context Server has to implement a specific interface, but is not limited concerning its data representation or internal structure. Context Servers register with an *Area Service Register*, that is part of the *Federation Tier. Federation* or *Nexus Nodes* broker between applications and Context Servers. Such nodes analyze application queries, determine which Context Servers are able to answer the query, forward the (partial) queries to these servers, compose the answers into a consistent view and return it to the application. Nexus nodes may also provide so-called value-added services that make use of the federated model. One example is the combination of basic events into complexer ones: A service monitors the position of persons on a friend-list and fires an event if two of them meet [DR03]. The whole system uses location as a primary modeling dimension. Large-scale models of streets and buildings with a backing relational database are as feasible as small-scaled sensor systems. The large scale and high level of detail demand a sophisticated computing infrastructure to provide the necessary resources.

CoBrA The *Context Broker Architecture* [CFJ03] provides clients with context information in localized *smart spaces* like intelligent meeting rooms. A hierarchical system of computers maintains a shared model of the current context, effectively providing a knowledge repository for clients within the space. The architecture is assumed to be always accessible, which relieves resource-limited devices of the responsibility to store and process complex context information. Clients entering a smart space send a URL with their own policy to the room's broker. The broker then receives the policy and shares the contained data with other clients. CoBrA also covers privacy aspects. Clients can specify detailed rules about the usage of their data. The use of a stationary infrastructure alleviates the management of complex context models, but limits the quality of information. Only facts about clients registered with the system can be regarded as reliable.

P2PC The Peer-to-Peer Coordination Framework [CF03] specializes on supporting mobile ad-hoc environments with frequently changing devices. The framework contains a middleware that handles the automatic discovery of other devices and the sharing of context. Context is perceived as a collection of attributes describing a situation. The authors do not give an example or proof of feasibility, but experiences from the framework influenced the later proposal of P2P patterns [FHM$^+$08].

MUSIC In [RWK$^+$08] the authors present a context model as a result of the European IST project called *Self-Adapting Applications for Mobile User in Ubiquitous Computing Environments* (MUSIC) [MUS07]. It incorporates many special characteristics of context such as incompleteness, ambiguity, uncertainty and temporal aspects. The decisive difference to other approaches is the use of *Model-Driven-Development* (MDD) to facilitate the *ease of development*. Three layers are defined in their architecture: The *Conceptual*, *Exchange* and *Functional* layer. The exchange and functional layers are similar to previous approaches. The exchange layer deals with exchanging context data in various formats like XML, *JavScript Object Notation* (JSON) or *Comma Separated Values* (CSV), whereas the latter refers to the actual implementation of the model's representation in different languages like Java or .NET. The conceptual layer consists of a context meta-model defined in UML and a context ontology in OWL. In the meta-model context information is abstracted by *context elements* which may be composed of *context values* and corresponding *metadata*. Elements, values and metadata correspond to a *Dimension* (cf. Fig 5.16). Thus, different *context scopes* can be determined that cluster context information belonging to the same domain. For example, the group "Location" could comprise values like "Longitude", "Latitude", "Room number" and "Accuracy". The meta-model is complemented by the MUSIC context ontology. It uses a two-level hierarchical approach in which a top-level ontology is refined by a more specific underlying ontology. The inclusion of a meta-model and usage of MDD enables the automatic generation of objects holding context information. As different representations are formally defined, transformations and automatic equivalence checking is possible.

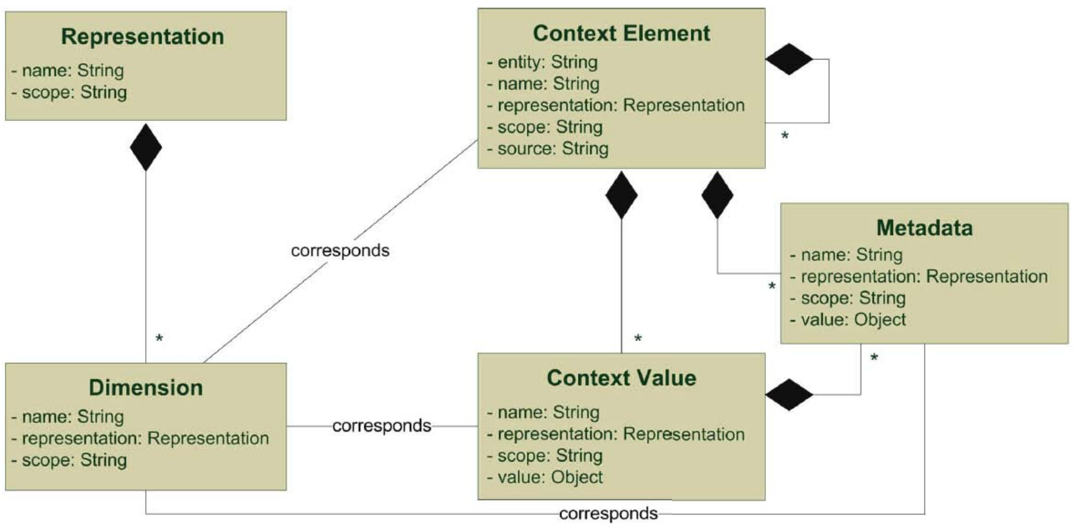


Figure 5.16.: The MUSIC context meta-model [RWK+08]

5.5. Summary

The decision, which context modeling technique should be used, has to be based on the demands of the ambient intelligent system. This section summarizes the identified modeling approaches. The following list states and justifies criteria for the selection. It most widely follows the argumentation by Strang et al. [SLP04].

Distributed Composition Ubiquitous systems lack a central instance which is responsible for managing context data. The ideal context model facilitates the distributed storage and composition [SLP04].

Partial Validation As a result of the distributed composition, there will probably be no point in time and space where the context model is completely and consistently stored on one single device. It is therefore necessary to be able to partially validate the model on both the structure and on the instance level [SLP04].

Richness and Quality of Information The accuracy and quality of sensor data and derived high-level context is subject to changes over time. The model has to include quality and richness indication [SLP04, EHL01].

Incompleteness and Ambiguity At any given point in time context data is usually incomplete or ambiguous. This is often the result of sensor inaccuracy. Context models should provide mechanisms for handling incomplete or ambiguous data [SLP04, HI04].

Level of Formality Context facts and relationships have to be stored in a precise and traceable manner. A high level of formality supports this [SLP04].

Tool Support For developers to make full use of the context model, software tool support is required. This is especially true if large models are created, because software tools can be used for validating the model.

Performance A small memory footprint and resource-efficient reasoning is desirable for any context model. High demands in this part may prevent the context model's use on devices with low memory and computing power.

Context History Previous user decisions can be used to predict his likely future needs [MJM09]. The modeling technique has to make it as easy as possible for developers to store and access previous context information.

Table 5.1 summarizes the six modeling approaches from Section 5.1. As with most requirements, they can never be completely fulfilled at the same time. There are always trade-offs between different requirements. For example, high performance can only be achieved by constraining the expressive power in terms of the richness and incompleteness requirements. A high level of formality usually leads to better machine-readability at the cost of human-readability.

The *Key-Value* approach is the simplest and most performant modeling technique, but also needs the most implicit knowledge as the structure itself imposes no semantics to the data saved. As there is no concept of annotating the key-value pairs, quality, incompleteness and history are hard to model. Structure is an advantage of *markup languages* like XML. They are very flexible and supported by many software tools and platforms which makes them ideal for the use in distributed environments. The partial validation is possible due to the possibility to include schema definitions. Still, the ambiguity and incompleteness handling cannot be considered intrinsic to the model and have to be handled by the application. *Graphical approaches* are the most intuitive way of modeling. Their strength lies in supporting human structuring processes. However, to be used automatically, they have to be serialized in a machine-readable format, which is usually again a markup language. They also require a specialized software tool to be used efficiently. *Object-oriented* context models have a sufficient level of formality and are well suited for the use in distributed environments. The tool support depends on the programming language used, but is usually adequate. Validation is done by compilers and runtime environments. Problems arise if different programming languages are used in a single environment. This heterogeneity implies the use of standardized serialization formats. Models based on *logic* are the most formal approach and mostly used in AI-systems. Their major problem is their poor performance and high resource demand. *Ontologies* are a very powerful and formal modeling approach. With the standardization of the Web Ontology Language (OWL), software tools have become available for modeling and reasoning. This approach facilitates the building of shared knowledge as extensions to the vocabulary are relatively easy. As with logic-based models, the reasoners are complex and high computing power is necessary.

Location models can contain *symbolic* and *geometric* coordinates. Out of the discussed symbolic models the combined (graphical and set-based) approach provides most flexibility. Queries for distances, nearest neighbors and spatial containment are possible on a qualitative level. To further improve the level of detail and to provide greater accuracy of distance calculations the symbolic models can be extended with geometric coordinates. The resulting *hybrid* model allows the use of arbitrary geometric area definitions. Partial subspaces provide a variable level

	Key-Value	Markup	Graphical	Object-oriented	Logical	Onto-logical
Composition	-	+	-	++	++	++
Validation	-	++	-	+	-	++
Quality	−	-	+	+	-	+
Ambiguity	−	-	-	+	-	+
Formality	-	+	+	+	++	++
Tools	-	++	-	++	−	+
Performance	++	++	*	++	−	−
History	−	-	+	+	+	++

Legend	
++	The modeling approach fully supports this requirement
+	The modeling approach partially supports this requirement
-	The modeling approach hinders this requirement
−	The modeling approach does not support this requirement
	Needs to be transformed into another representation before automatic processing

Table 5.1.: Different modeling techniques' degree of fulfillment concerning context requirements

of detail while the constraint of normal subspaces (all symbolic coordinates must also have known geometric extends) leads to models in which exact and quantitative statements about the spatial relationships between all coordinates can be made.

The Context Toolkit and Hydrogen frameworks have been analyzed in [BDR07]. All three frameworks aim at supporting a broad spectrum of context-aware applications. The *Context Toolkit* is a widget-based, distributed framework that makes high use of encapsulation and abstraction principles, but lacks a discovery mechanism and handling of disconnections in an unreliable network. *Hydrogen* runs completely on the context-aware device and compensates for connection losses with local data. Unfortunately, mainly local sensors are considered and the sharing of (possibly ambiguous) context data is an open issue. The *Java Context Awareness Framework* is well-tested and includes access control mechanisms. The communication benefits from the use of Java libraries and serialized objects, but interoperability with other programming languages is abandoned.

6. Context Model

Before describing the implementation and details of the context engine, this chapter formally introduces the *context model* which is based on XML. The whole schema can be reviewed in Appendix A.1. XML was chosen because of its interoperability, the available tool-support and its expressive power on the structural level. Furthermore, object-oriented handling of the XML tree is possible with the help of XMLBeans, which will be discussed in Chapter 7. The context model does not claim to follow a generic modeling approach. The focus of this thesis is on the development of the necessary infrastructure to handle context in the SmartMote, not on providing a universal context (meta-) model. Therefore this model focuses on *location* as a primary context type. It complements the interaction zone definitions in the RUM which are only defined by symbolic coordinates. While not being merged in a single model, the combination of RUM and context model provides all benefits of a *hybrid* location model as described in Section 5.2. After stating the requirements the individual context elements are discussed.

6.1. Requirements

The context model has to fulfill the following requirements:

(R1) It must be able to store geographic location information.

(R2) Objects of interest are: one single SmartMote and an arbitrary number of Ubisense and NFC tags. The positions of these objects must be modeled.

(R3) A history of last known positions of objects must be maintained. The history may be limited to a number of newest entries. Entries may contain an optional timestamp attribute.

(R4) The model must be able to store different kinds of zones and their geometric extents.

Two restrictions must be noted. First, the model considers only a single room. This assumption is also made by the SmartMote in general and fits the conditions of the *SmartFactory*$^{\mathbf{KL}}$, in which all modules are located in one hall and on the same floor. If more rooms should be added, a transformation of the context model's coordinates is necessary. Second, the number of SmartMote devices is also limited to one. Multiple UCDs have not been tested in the *SmartFactory*$^{\mathbf{KL}}$ but may be an aspect of future work.

6.2. Model Details and XML Structure

Figure 6.1 shows the top levels of the model's XML schema. The root element is of the XML complex type `TContext`, which is subdivided into two second-level types: `TObjects` and `TPlaces`. The following sections describe each XML type in more detail starting with points as basic geometric entities.

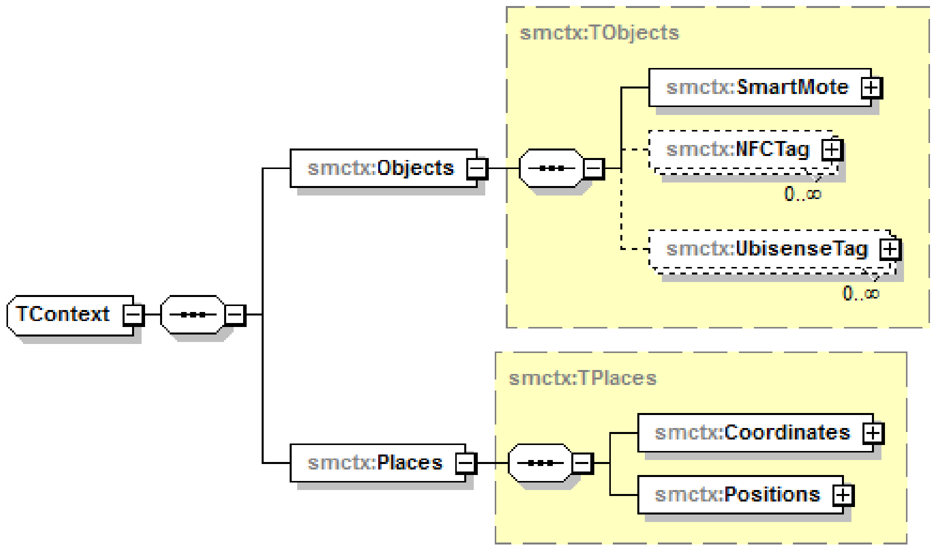


Figure 6.1.: Root element with children TObjects and TPlaces

6.2.1. Points

Points (cf. Figure 6.2) are represented by the XML type `TPoint`. They are the most basic modeled geometric entity. The three coordinates `X`, `Y` and `Z` are absolute cartesian coordinates with the Ubisense coordinate system being the reference system. Points are the only types in the context model containing definite values for coordinates. Points must also have a unique `id`, so that other elements may refer to them. This has two advantages: First, it removes redundancy from the model and second, it facilitates the transformation of all coordinates if a new reference system is introduced, because they are clustered at one place. Points also save a list of references to other places in which they are contained in their `containedInCoordinates` attribute. This list is used by the reasoner to determine the currently active interaction zones.

6.2.2. Object Positions and Position History

The `TPositionHistory` (cf. Figure 6.3) depicts the "position memory" of an object. Its children nodes are references to existing points. The optional attribute `maxItems` contains an upper limit for the length of the list and therefore determines the length of the memory. The reasoner

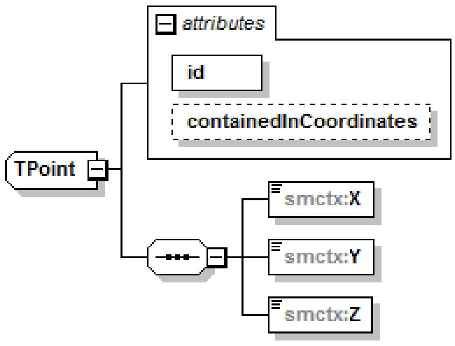


Figure 6.2.: XML type TPoint

contains a rule that deletes the oldest list elements if a new one would exceed the limit (cf. Section 7.4.3). The references are of the type `TPointReference` and contain a timestamp and the id of the referenced `TPoint` instance in their attributes `time` and `pointID`.

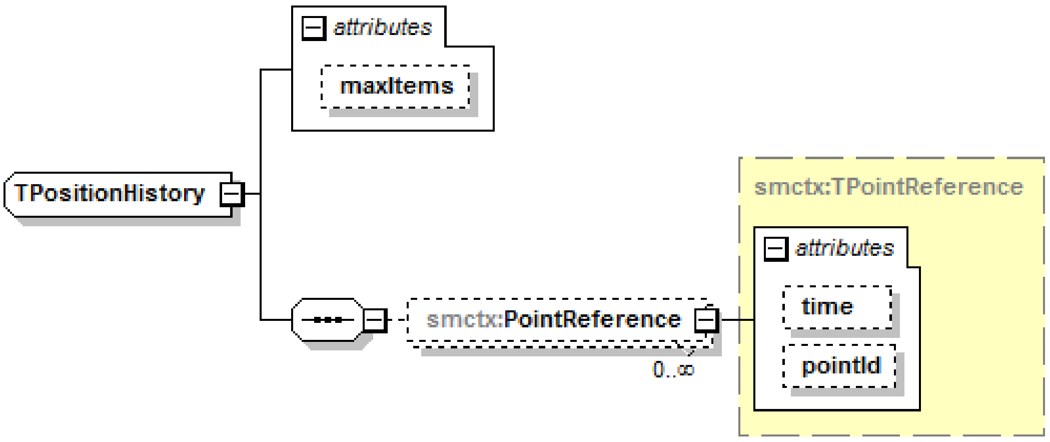


Figure 6.3.: XML types TPositionHistory and TPointReference

6.2.3. Objects

In accordance with the second requirement (R2), the `TObjects` type holds one `TSmartMote` and two lists of `TUbisenseTag` and `TNFCTag` children with arbitrary size (cf. Figure 6.5). Each of these types holds one position history as described above. The remaining individual features of each type are discussed below.

SmartMote

In addition to the position history, the SmartMote has two attributes: `id` which contains its unique identifier and `ubisenseTagID` which contains a reference to a Ubisense tag. The underlying assumption is that the SmartMote device is attached to exactly one Ubisense tag (cf. Section 7.1). To determine the SmartMote's position, the reasoner has to know the `id` of the tag.

Ubisense Tags

Ubisense tags may have an additional non-unique designation in their `name` element. This information does not need to be modeled at design-time, because the Ubisense system provides it together with the tag's position.

NFC Tags

NFC tags store further information about the factory module they are attached to. The records contain strings following the data structure described in [Kha09]. The fields are: `Firma` (company), `Geraetetyp` (device type), `Geraetetyp_Id` (device type id), `Geraetenummer` (device number), `Geraete_Id` (device id), `Geraete_Name` (device name), `Modul_Name` (module name) and `Modul_Nummer` (module number). For simulation purposes a minimalistic editor has been implemented in Java that can store and retrieve the records from NFC tags (cf. Figure 6.4). Currently, MiFare [MIF25] cards with 1 or 4 kB space are used as tags, but the card reader and software are able to handle other types as well. One exception concerns the position history of NFC tags. As their position is static and previously known, the histories of NFC tags must contain a valid point reference from the beginning. Only the timestamp is updated during run-time and tags without known position are ignored by the reasoner.

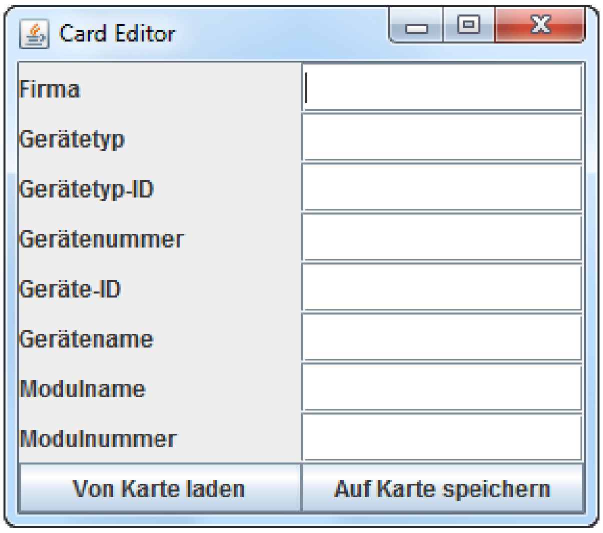


Figure 6.4.: Minimalistic editor to write and read NFC tags

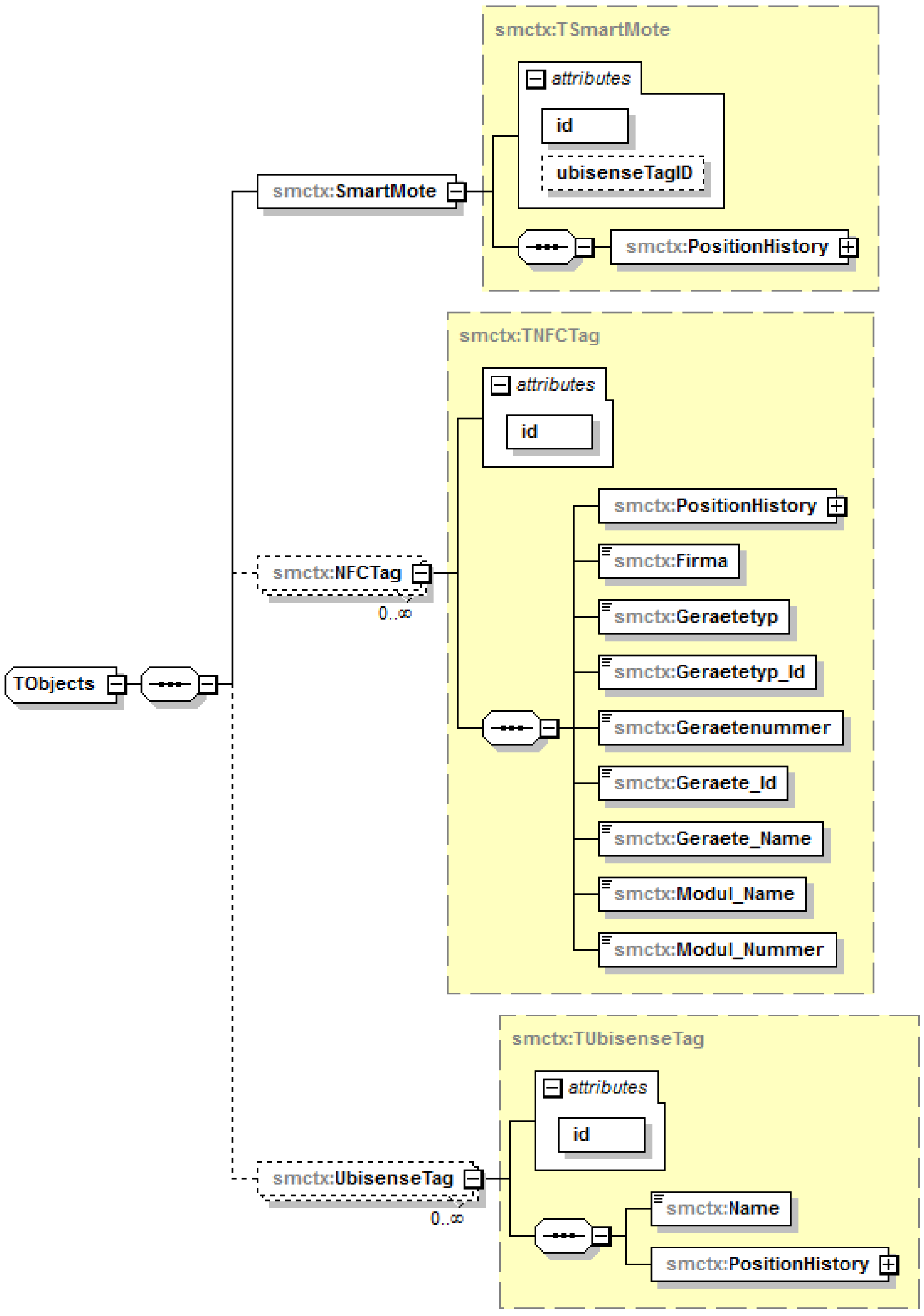
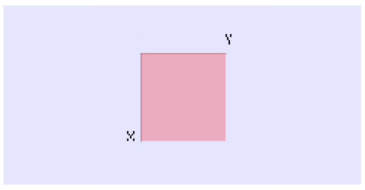


Figure 6.5.: XML types TObjects, TSmartMote, TUbisenseTag and TNFCTag

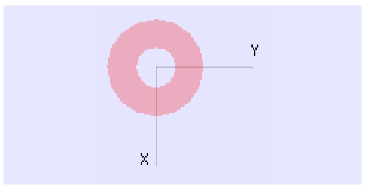
6.2.4. Places and Zones

Geometric positions and zones are modeled as children of `TPlaces` (cf. Figure 6.7). `TPoints` is simply composed of points as described above. Geometric zones (i.e. the geometric extents of interaction zones) are modeled as children of `TCoordinates`. Currently, the context model distinguishes three different types of zones: Blocks (`TBlock`), rings (`TRing`) and arcs (`TCircleSegment`). The zones are defined as basic two-dimensional shapes with a height value instead of three-dimensional objects. This is consistent with the scenario as factories (and buildings in general) are usually divided into floors. As zones in the context model are referenced by the interaction zones in the RUM, a unique identifier for each one is mandatory. Figure 6.6 shows an example of how the zones are defined in the model and what they look like. The types in detail are:

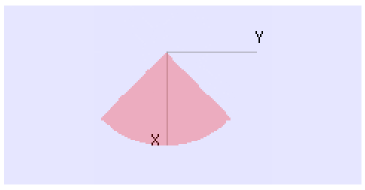
- *Blocks* are based on a simple rectangular basis. They are defined by two points which must have the same z-coordinate and a height. Blocks are already sufficient for many purposes, because the Ubisense coordinate axes are aligned according to the modules.
- *Rings* have a source point, an inner and an outer radius. They offer the best use to model range-dependent behavior.
- *Arcs* are defined by a starting- and endpoint as well as an opening angle. Arcs are the best solution to model perspective-dependent behavior, for example if a screen must be watched that gets too blurred if the angle is too high.



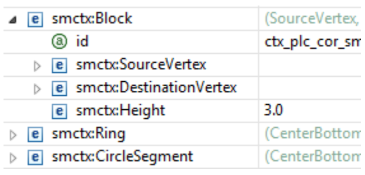
(a) Block



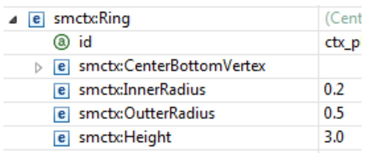
(b) Ring



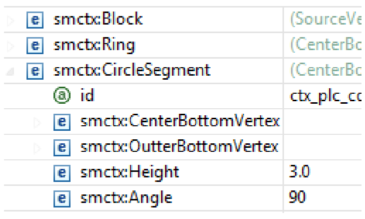
(c) Arc



```
◢ e smctx:Block                 (SourceVertex,
    a id                        ctx_plc_cor_sm
  ▷ e smctx:SourceVertex
  ▷ e smctx:DestinationVertex
    e smctx:Height              3.0
▷ e smctx:Ring                  (CenterBottom
▷ e smctx:CircleSegment         (CenterBottom
```



```
◢ e smctx:Ring                  (Cent
    a id                        ctx_p
  ▷ e smctx:CenterBottomVertex
    e smctx:InnerRadius         0.2
    e smctx:OutterRadius        0.5
    e smctx:Height              3.0
```



```
  e smctx:Block                 (SourceVe
  e smctx:Ring                  (CenterBc
  e smctx:CircleSegment         (CenterBc
    a id                        ctx_plc_cc
    e smctx:CenterBottomVertex
    e smctx:OutterBottomVertex
    e smctx:Height              3.0
    e smctx:Angle               90
```

Figure 6.6.: Three types of interaction zones and their representation in the context model

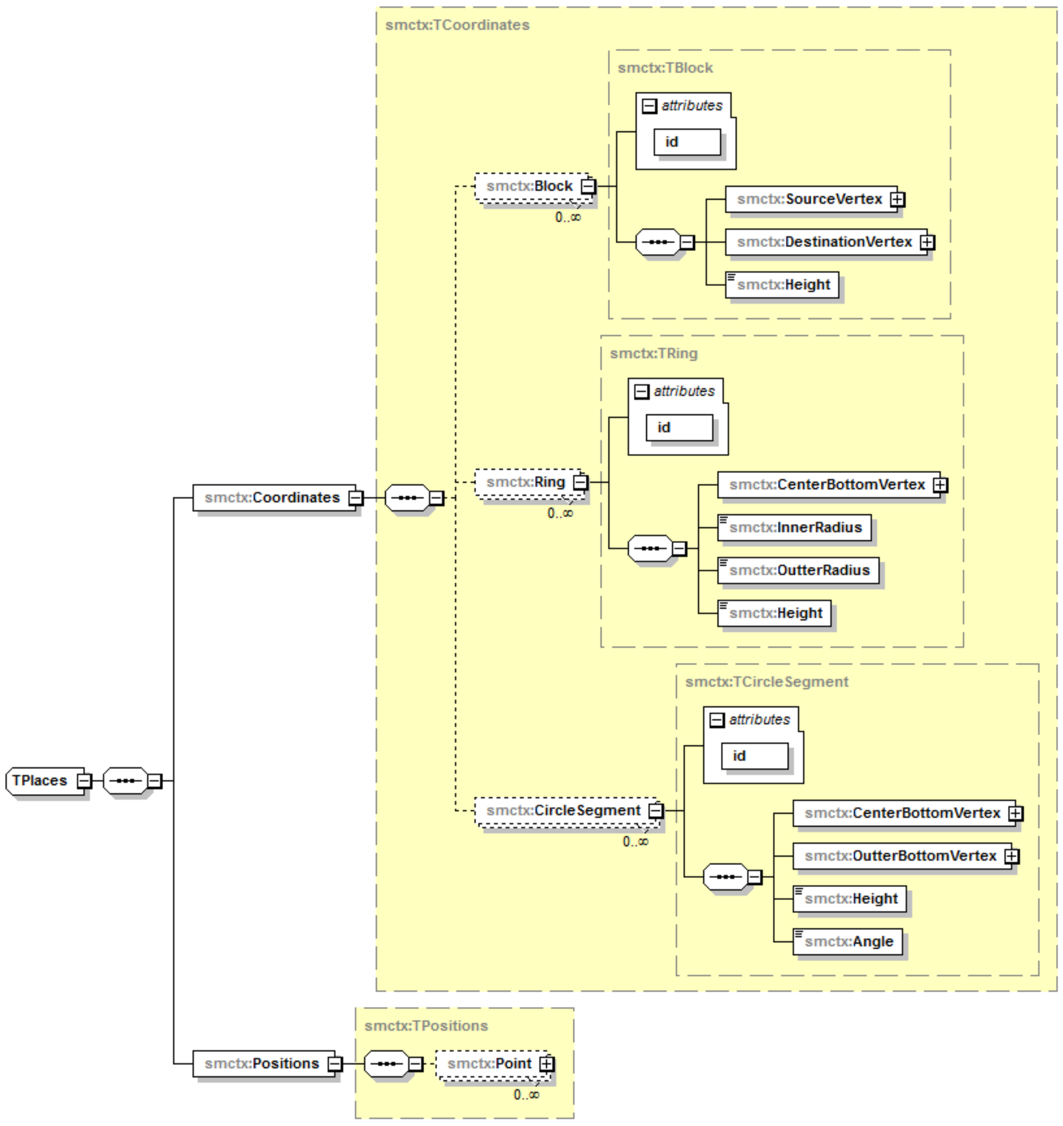


Figure 6.7.: XML types TPlaces, TPositions, TCoordinates, TBlock, TRing and TCircleSegment

7. A Context Engine for the SmartMote

This chapter introduces the developed *context engine* for the SmartMote. After describing the *setup*, *requirements* are stated which the individual components should satisfy. The architecture, software approaches used and implementation details form the rest of the chapter.

7.1. Setup

This section describes the hard- and software components used. As an abstract RUM of the *SmartFactory*$^{\mathbf{KL}}$ has already been created in [Gö9], the different modules and properties of the devices in the factory are not specifically mentioned.

Hardware

The SmartMote currently runs on a "Slimbook P230" [Pac22]. This TabletPC is equipped with integrated Bluetooth and WLAN which makes it ideal for mobile use. It also has sufficient computing power to run the Java-based software and therefore provides a good compromise between screen size, weight, battery power and performance. Its touch-sensitive screen can be operated by simple finger touch or with the help of a stylus.

Unfortunately, the SmartMote is not equipped with an RFID or NFC reader. As tags for both systems are installed at fixed locations in the *SmartFactory*$^{\mathbf{KL}}$ (cf. Section 2.2) and should be used as location input, an external reader is used. The "ACR122" NFC reader [Adv22] provides NFC functionality via a USB cable. At the moment, RFID sensing is not implemented, but the integration of RFID at a later time works analogously to the NFC system.

For the three-dimensional sensing of locations, a commercial ultrasonic positioning system is used. The Ubisense [Ubi22] sensors are integrated into the factory infrastructure. The position of Ubisense tags is sensed continuously and can be acquired from a web-service over the permanently available WLAN. Appendix A.2 shows an example XML file that is returned by the webservice. Each Ubisense tag has a unique identification number with the help of which its record in the XML tree can be found.

Until alternative hardware with integrated NFC, RFID and Ubisense capability is available, the tags and the reader have to be externally attached to the TabletPC. This is a drawback of the current demonstrator, but has no influence on the underlying principles.

Software and Development Environment

The original SmartMote software and the introduced context engine are written in Java version 1.6 [Jav22]. The eclipse IDE [Ecl22] was used for editing Java sourcecode and all necessary XML schema and model files. For accessing the NFC reader in Java, the JPCSC library [Lin22] is used. The run-time access to XML data is provided by Xmlbeans [Apa22], which binds types from an XML schema to Java types. More details on this can be found below.

7.2. Software Requirements

With the available hard- and software in mind, requirements for the context engine can be presented. They are ordered component-wise, which also corresponds to the engine's Java package structure.

Ubisense Sensors

(R5) The Ubisense sensor system has to poll the Ubisense service over WLAN at least once per second.

(R6) If data has been retrieved, it must be parsed and stored in the model together with the time of retrieval.

NFC Sensors

(R7) The NFC sensor system has to connect to to the NFC reader, establish a handle and wait for an NFC contact.

(R8) If an NFC tag is read, the data must be parsed and, if it matches the NFC data scheme (cf. Section 6.2.3), be stored in the model together with the time of retrieval.

7.2.1. Context Model

(R9) The model component must maintain an instance of the XML schema.

(R10) It must also provide suitable access mechanisms for clients.

(R11) A notification system must be implemented that notifies registered listeners of changes in the model. Clients must be allowed to subscribe to changes on specific (existing) data and to all current and future values of a specific type (type-based subscription).

7.2.2. Reasoner

(R12) The reasoner has to provide a set of rules that operate on the model.

(R13) It must be ensured that rules can be added and removed at run-time without compromising the model's integrity.

(R14) A rule must be implemented that derives the position of the SmartMote from available sensor data. A statistical process should correct possible inaccuracy with the help of the last known positions.

(R15) The reasoner must maintain a list of currently active interaction zones. SmartMote position changes have to trigger an update of this list.

(R16) If the list of active zones changes, the RUM has to be filtered accordingly and and the GUI generator has to be notified of the change.

7.3. Architectural Overview

Figure 7.1 shows a conceptual overview over the the context engine. The whole system is sensor-driven, which means that the arrival of new sensor data triggers all subsequent actions. The following steps are an example sequence of events that are triggered by an NFC contact:

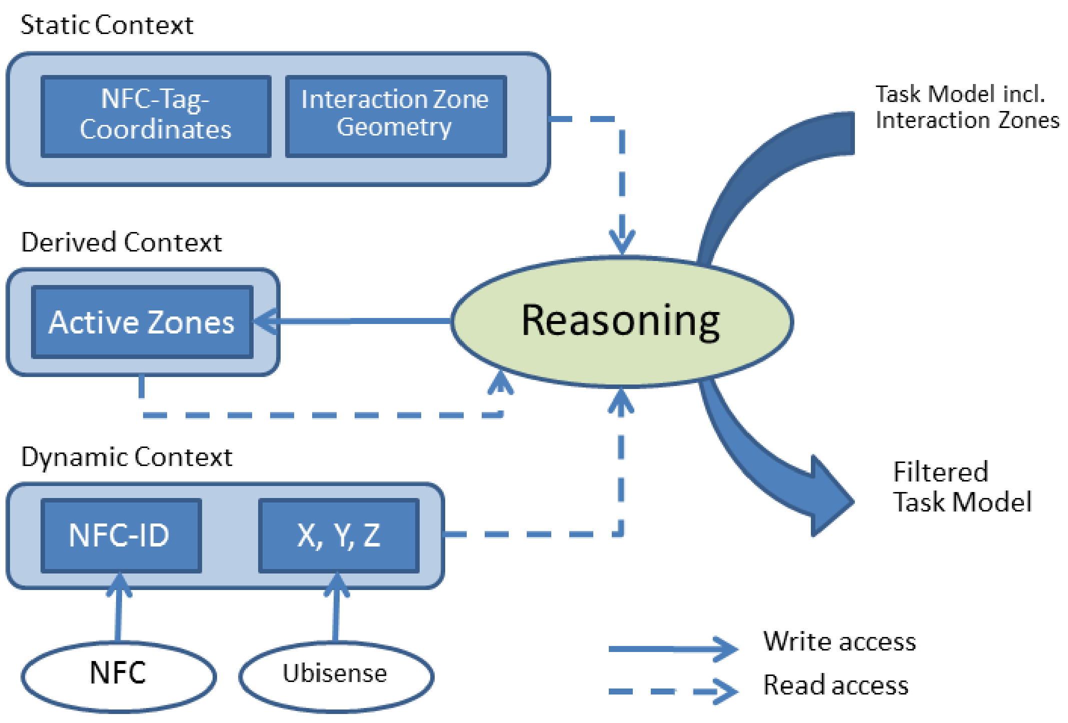


Figure 7.1.: Conceptual overview over the context engine

1. The NFC reader recognizes an NFC tag.
2. The sensor component reads the data contained on the tag and checks if it has the correct type and structure. After parsing the data, the sensor component updates the tag's corresponding record and timestamp in the model.

3. The new position entry triggers a rule in the reasoner. The rule determines that the SmartMote position should be updated with the new position. It fetches the updated tag's location (static context) and writes a new entry into the SmartMote's position history.

4. Another rule is triggered that listens for new positions in general. It examines coordinates and compares them to the interaction zones. It attaches a list of active zones to the positions entry.

5. The reasoner is notified and fetches the list of active zones. If it differs from the last one, the RUM is filtered according to the new set of zones. This leads to a GUI update.

7.4. Components

After giving a general overview over the engine a detailed description of each individual component is given. Figure 7.2 shows a UML diagram of the core classes. The complete package and class list can be reviewed in Appendix A.3.

The GUI generation process from the task model is described in [Mas08]. Its main interface to the context engine is the `DataManager` class in the package `agse.gabigui.data`. This class manages the RUM representation. The `Reasoner` hooks into the `DataManager` and fetches the RUM object[1] Internally the reasoner manages two instances of the RUM. After acquiring the context-sensitive RUM from the `DataManager` the reasoner makes a copy, on which it applies its set of filters. If the filters change at a future point in time, not the whole RUM has to be transformed again, but only the changed parts. The resulting context-insensitive RUM is then sent back to the `DataManager`.

7.4.1. Model

The model is included in the `agse.gabigui.context.model` package. The main class is the `ContextModelManager`. Before discussing its implementation, this section starts with the description of XMLBeans and how it is used to get an object-oriented representation of the XML tree.

The XMLBeans Approach

XMLBeans is a technology for binding XML types to Java. Figure 7.3 illustrates its concept. The *Schema Compiler* (scomp) generates a set of Java interface- and class-definitions. This type system can then be used to derive an object tree that corresponds to the schema. The generated classes possess get- and set-methods for their contents, attributes and children. The object representation can be serialized to or loaded from an XML document. The Java type system can also be used to validate both representations, which occurs automatically when loading a document. These operations can also be performed on single elements and parts of the XML

[1]XMLBeans is also used with the RUM, which means fetching the *RaumbasiertesBenutzungsModell* class as root element is sufficient to get access to the whole task model.

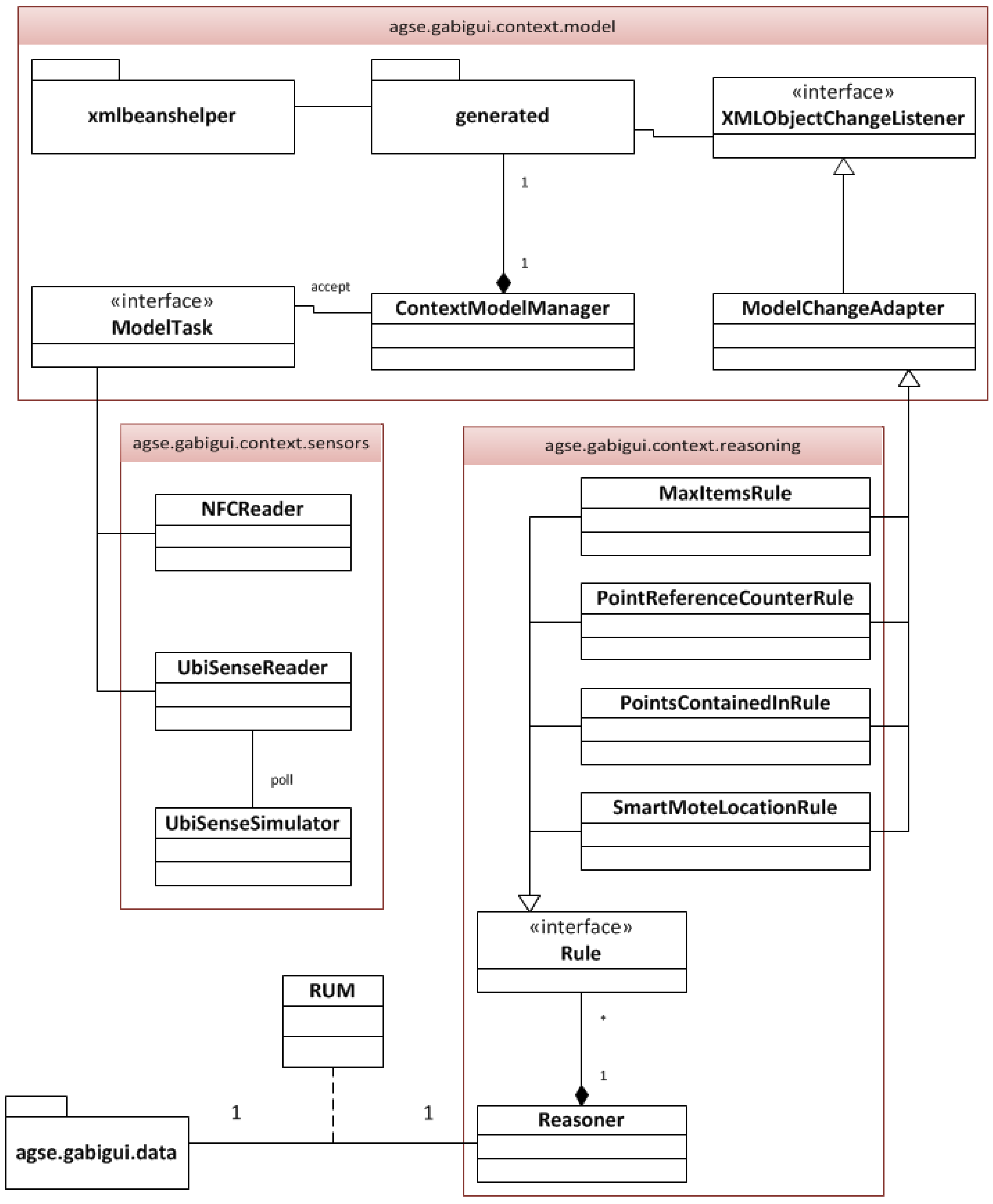


Figure 7.2.: Core classes of the context engine

document. Another advantage is that the object tree can be queried via the XPath and XQuery standards. The results of such queries are Java objects. This approach therefore combines the advantages of markup languages and object orientation as described in Chapter 5. The generated classes and interfaces are contained in the `agse.gabigui.context.model.generated` package. This fulfills requirement **(R9)**.

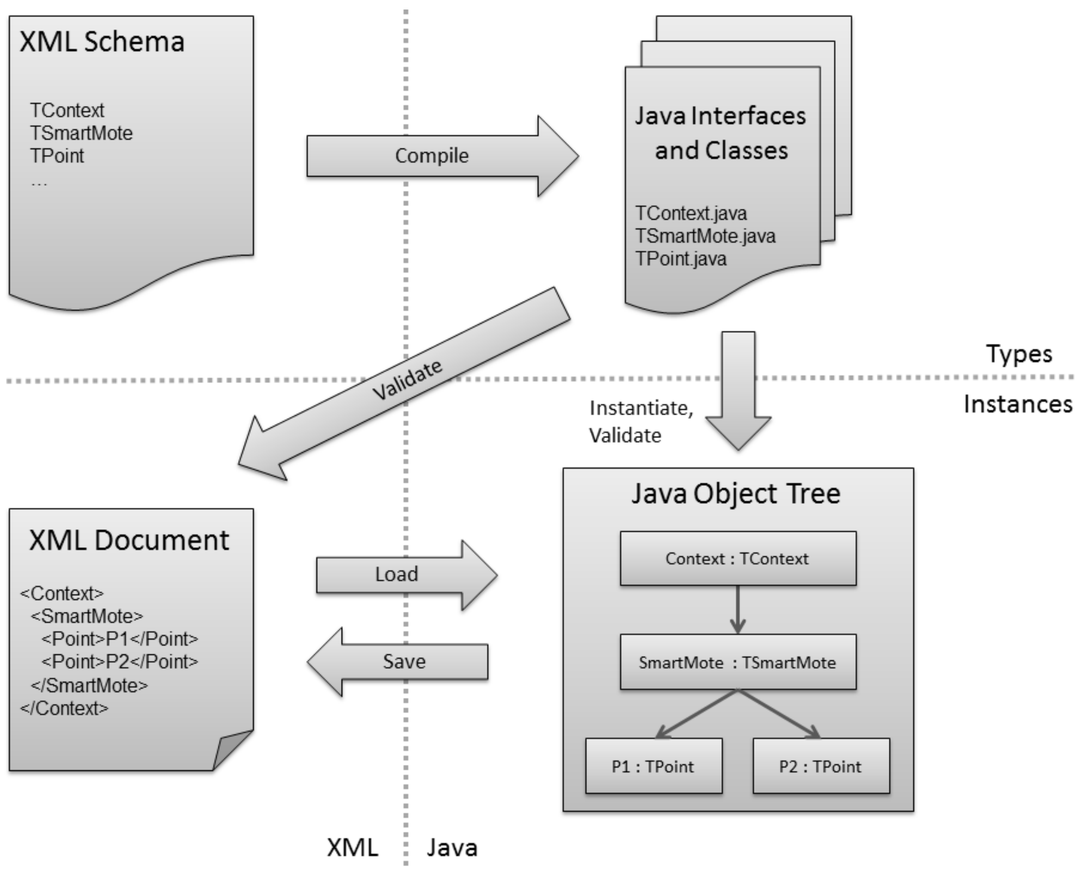


Figure 7.3.: XMLBeans approach: A compiled schema is the basis for object-oriented processing.

Observer Extension

Requirement **(R11)** poses a problem for XMLBeans. The generated Java classes do not implement any notification or callback mechanisms. Access to the model works on a query-basis only. To implement the required observer pattern functionality [GHJV09], additional effort is required. Fortunately, XMLBeans provides two extension mechanisms which can be used to add the functionality needed. First, custom methods can be added to the generated interfaces. During run-time a call to such a method is forwarded to a static helper class with the source object as an additional parameter. The `XMLBeansChangeEmitter` interface is shown below.

```
public interface XMLBeansChangeEmitter {
  public void addModelChangeListener(
      XmlObjectChangeListener modelChangeListener);

  public void removeModelChangeListener(
      XmlObjectChangeListener modelChangeListener);

  public boolean hasModelChangeListeners();

  public void fireModelChangeEvent(TContext model, XmlObject source,
      int action, boolean isAttribute, QName name, Object oldValue,
      Object newValue);
}
```

All objects generated by XMLBeans implement this interface. It is therefore possible to register `XmlObjectChangeListener`s on the model values. The second feature is called *PrePostSet-Feature*. Whenever a value of an object is changed by a call to one of its set-methods, a static helper method is notified. This happens at the beginning of the set-method and at the end. By comparing the object's state, the helper method can calculate the exact changes and call the `fireModelChangeEvent` on the source object which then notifies all registered listeners. The type-based subscription mechanism is also implemented with the help of this feature. All necessary extensions are encapsulated in the `agse.gabigui.context.model.xmlbeanshelper` package. For easier handling of the observer functionality, the class `ModelChangeAdapter` has been created. Clients extending this class can register themselves to values in the model and then get notified of changes.

```
public class ModelChangeAdapter implements XmlObjectChangeListener {
  public void nodeCreated(TContext model, XmlObject source, QName name,
      XmlObject newNode) {
  }

  public void nodeDeleted(TContext model, XmlObject source, QName name,
      XmlObject oldValue) {
  }

  public void nodeUpdated(TContext model, XmlObject source, QName name,
      Object oldValue, Object newValue) {
  }

  public void attributeSet(TContext model, XmlObject source, QName name,
      Object newValue) {
  }

  public void attributeUnset(TContext model, XmlObject source, QName name,
      Object oldValue) {
  }

  public void attributeUpdated(TContext model, XmlObject source, QName name,
      Object oldValue, Object newValue) {
  }
}
```

The `XMLBeansHelper` class also contains a static main method which starts the compilation with all necessary parameters for the extension. It is therefore not necessary to download and run the command line program *scomp* if the XML schema changes. Users can start the compiler from within the eclipse IDE. The generated Java source files are automatically saved to the `agse.gabigui.context.model.generated` package.

Access and Synchronization

The last open requirement is **(R10)** and concerns the access to the model. While it would be possible to grant clients concurrent access to the model, it is not advisable. XMLBeans is thread-safe, but only on an atomic level. Listing 7.1 shows an example of adding a new point to the list of positions. After line 2 the new point instance exists and can be accessed by other threads. At this point, the `id` attribute is not yet set and the X, Y and Z fields contain the standard values of "0.0". This can lead to inefficiency if derived values have to be recalculated or even worse, to faulty behavior.

```
1  TPositions pos = getPositions();
2  TPoint point = pos.addNewPoint();
3  // The new point now exists but has only default values
4  point.setId("point1");
5  point.setX(1.0d);
6  point.setY(1.0d);
7  point.setZ(0.5d);
```

Listing 7.1: Adding a new point

The solution is provided by the *Command Pattern* [GHJV09]. Instead of executing code directly on the model (cf. Figure 7.4(a)), clients create a `ModelTask` that contains the code and place it in the `ContextModelManager`'s queue (cf. Figure 7.4(b)). The manager fetches individual tasks from the head of the queue and executes them on the model. A new task is only fetched if the previous one finished its execution. The `ModelTask` interface is very simplistic:

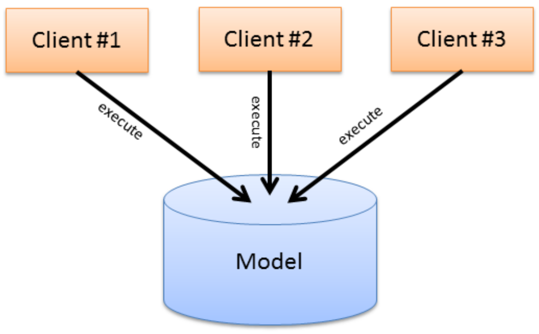
```
public interface ModelTask {

  public void executeOnModel(TContext model);

}
```

The `ContextModelManager` provides two methods for supplying tasks: The first places the tasks in the queue asynchroneously whereas the second also blocks until the task has been executed. It must be noted that it is still possible for listeners to get triggered during the execution of a task. Listeners may react to changes immediately or they can place a task in the queue themselves.

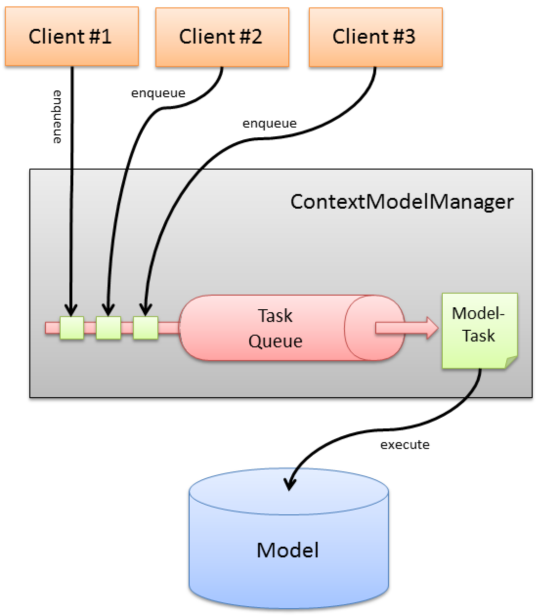
```
public class ContextModelManager {

  public static final void executeModelTask(final ModelTask task) {
    ...
  }

  public static final void executeAndWaitForModelTask(final ModelTask task) {
    ...
  }
}
```

7.4.2. Sensors

The `agse.gabigui.context.sensors.nfc` and `ubisense` packages contain the sensor components. The responsible classes are `NFCReader` and `UbiSenseReader` respectively.



(a) Clients simultaneously execute their code on the model



(b) Clients place their tasks in a queue, only one task at a time is executed

Figure 7.4.: Principle of synchronized access by using a command queue

Upon startup the `NFCReader` searches for available NFC hardware. Currently only the "ACR122" model is supported. After establishing an exclusive connection, the `NFCReader` waits for incoming NFC data. If an NFC tag is detected and data received, it is parsed according to the standard (cf. Section 6.2.3). Finally, a model task is created and placed in the model's task queue to write the new value to the model.

The `UbiSenseReader` is an active component that polls the Ubisense service once per second. The service is available via HTTP from a fixed URL, which can be specified in a global configuration file. Appendix A.2 shows an example of the delivered XML document. The XML document is parsed and if it contains new position data for a tag (`lastSeenOn` tag) a model task is created and scheduled for execution to update the tag's position.

7.4.3. Reasoner

The reasoner is located in the `agse.gabigui.context.reasoning` package. It mediates between the context engine and the GUI generator and handling both, the transformation of the RUM to its context-insensitive form and the operation on the context model. Aspects of interest are how additional rules can be added, which rules are already defined and how the transformation is performed.

Rules

Most rules extend or at least use a `ModelChangeListener` to react to changes in the model. Rules have to implement the `Rule` interface, which defines to methods:

```
public interface Rule {

  public void prepareModel(TContext contextModel);
  public void disconnectFromModel(TContext contextModel);

}
```

Rules can be added during run-time using the reasoner's `addRule` method. The `prepareModel` method is called as soon as the rule is registered and the reasoner is started (or already running). Rules are provided with a reference to the model and should add listeners to values of interest. They can also make use of the type-based subscription. As requirement **(R13)** states, rules should also be removable during run-time. This is ensured by the `disconnectFromModel` method. It is called if a rule is removed from the reasoner via its `removeRule` method and should de-register all listeners and opened connections to the model. Developers are currently required to do this manually. While this allows for a maximum of freedom, it also poses a risk if a developers forget to de-register some parts of their rules. In summary, the development of rules should be handled with care and tested thoroughly. Four rules have already been implemented:

MaxItemsRule This rule operates on all `TPositionHistory` elements whose `maxItems` attribute is greater than 2. If a child element is added to such a history and the number of children exceeds the attribute value the rule deletes the oldest entries from the history until the constraint is satisfied.

PointReferenceCounterRule This rule keeps track of points and references. It has a counter for each `TPoint` instance that indicates the number of references pointing to it. If the counter reaches zero, the point is deleted to keep the model lean. There is a trade-off between model complexity and CPU load. If a point is removed by this rule and added again at a later time, the list of zones that contain the point must be calculated again. In scenarios with lots of zones and objects, it might be more efficient to deactivate this rule and keep the points in the model.

PointsContainedInRule While the previous two rules were management rules to keep the model lean, this is the first rule that is needed for the RUM filtering. It is activated whenever a new point or a new zone is added or removed. It fetches the coordinates of affected points and the geometry of the zones to calculate the containment relations. This rule is responsible for keeping the `containedInCoordinates` attribute of `TPoint` instances up to date.

SmartMoteLocationRule The second filtering rule and most important one. The SmartMote-LocationRule determines the SmartMote's position. It uses a simple approach depicted in Figure 7.5. Basically, the NFC tag information is regarded as more accurate and has priority over the Ubisense location. If Ubisense information is used, the rule calculates the average of the last n values. This is a parameter that can be set in a global configuration file. A value of 3 has proven to be a good compromise between actuality and smoothing.

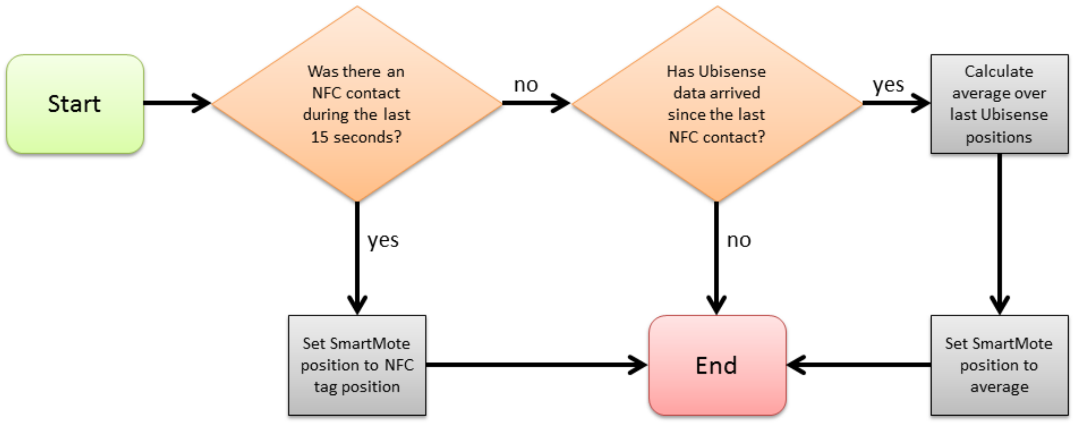

Start
Was there an NFC contact during the last 15 seconds?
no
Has Ubisense data arrived since the last NFC contact?
yes
Calculate average over last Ubisense positions
yes
no
Set SmartMote position to NFC tag position
End
Set SmartMote position to average

Figure 7.5.: Decision process of the SmartMoteLocationRule

Updating the RUM

The reasoner itself is registered as a listener on the SmartMote position history. If its location changes, the reasoner fetches the according `TPoint` instance and reads the list of zones in which the point is contained from its `containedInCoordinates` attribute. If the list differs from the one previously read, the RUM filter is updated. To do this, the reasoner has to know which interaction zone in the RUM corresponds to which geometric zone in the context model. This is done by using *Uniform Resource Identifiers* (URI) [BLFM23] to link from the RUM to the context model. `Bedienzone` (Interaction zone) nodes in the RUM have a child element `Koordinaten` (coordinates) containing an URI to link to external resources. Specifically, the reasoner searches these entries for a fixed prefix like “http://wwwagse.cs.uni-kl.de/smartmote/contextmodel/zones/”. The part following this prefix designates the `id` attribute of the geometric zone in the context model. After the reasoner identified the active zone of a `Geraeteverbund` (device compound) in the RUM, this zone’s use models become direct descendants of the `Geraeteverbund` as shown in Figure 7.6. It is also possible to define a default zone, that is picked if no other zone is active, using the keyword “DEFAULT” instead of an id. The transformation results in a RUM representation without interaction zones. Finally, the reasoner notifies the GUI generator of the changed RUM.

7.4.4. Simulation and Control GUI

For simulating the *SmartFactory*$^{\mathbf{KL}}$ environment the `UbiSenseSimulator` class has been implemented which is able to provide random position data to the `UbiSenseReader`. When polled for values, the simulator slightly changes the coordinates of the Ubisense tags to simulate a random path through the factory hall.

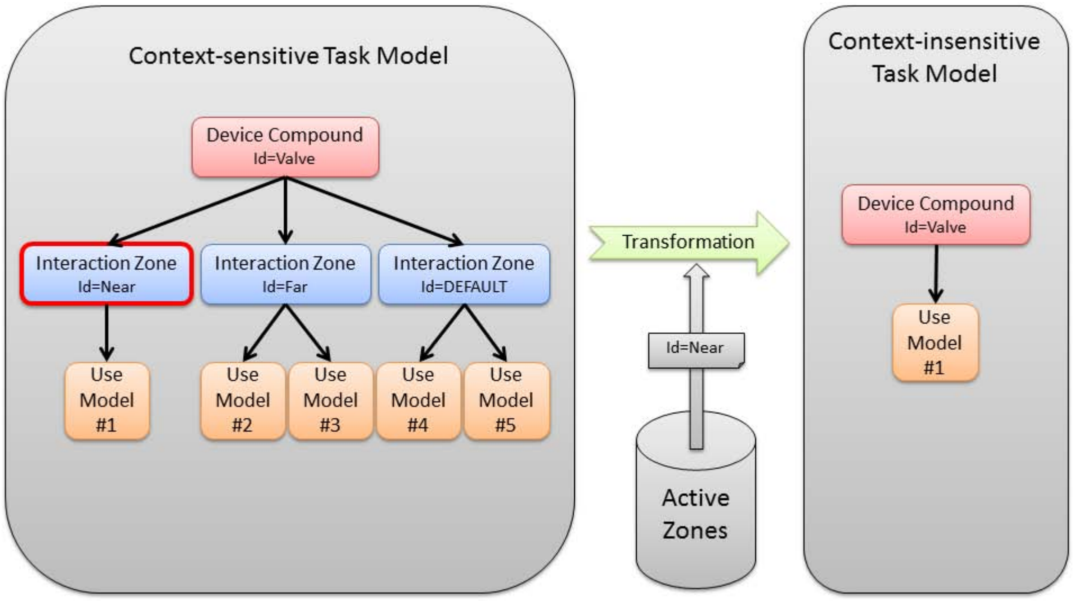


Figure 7.6.: RUM transformation: The children of an active interaction zone become direct descendants of the parent device compound. All other zones are discarded.

A control GUI has also been implemented for testing and visualizing the model's current values. It consists of two parts: An overview over the factory layout and an interface containing control elements for the modules, including the sensors and reasoner. Figure 7.8 shows the layout window. The black areas are covered by factory modules. The status label at the bottom shows the current pixel and Ubisense coordinates corresponding to the mouse cursor's position. Figure 7.7 shows the control window. The available features of the individual components are as follows:

- *View options*: The view options control the visual appearance of the layout windows and which components are shown.
  - *Show SmartFactory layout*: Shows or hides the layout window.
  - *Show point grid*: Overlays the layout with a point grid.
  - *Show points present in the model*: The `PointReferenceCounterRule` may remove point instances from the model if they are no longer referenced. With this option activated the layout view shows which points are currently contained in the model.
  - *Show SmartMote*: Shows the current SmartMote position as green circle.
  - *Show UbiSense*: Shows all Ubisense tags' current positions as blue circles.
  - *Show NFC tags*: Shows all NFC tags' positions as orange circles.
- *Ubisense*: This panel provides options for controlling the Ubisense sensor component.

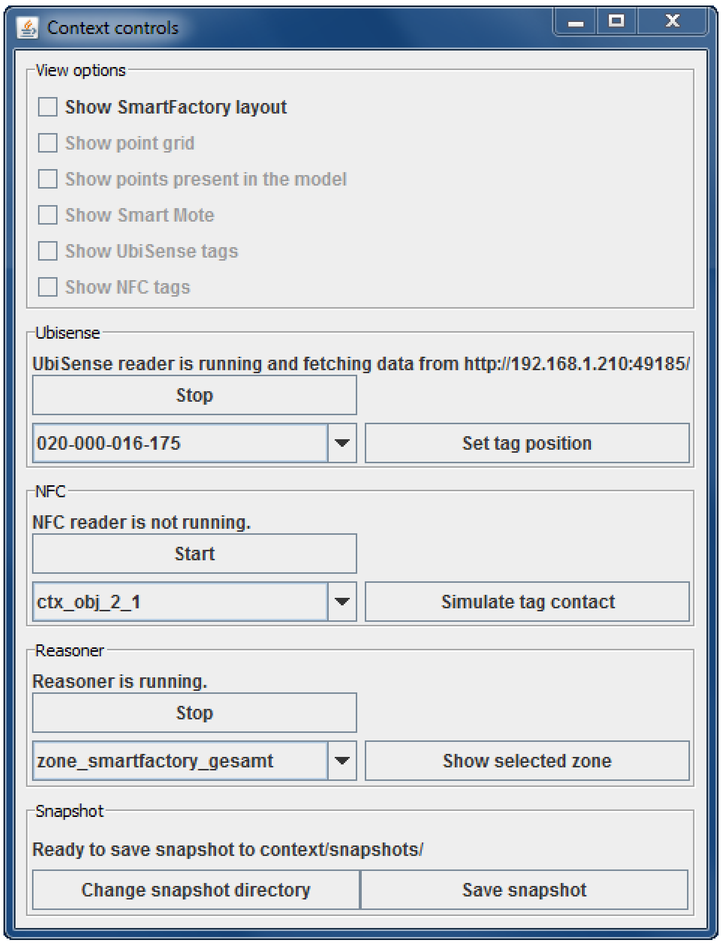
  - The *status label* indicates whether the Ubisense sensor component is currently running and which source it fetches data from.
  - The *start / stop* button activates or deactivates the Ubisense sensors. The NFC component and the reasoner have the same button and can be started and stopped during run-time.
  - A list of all available Ubisense tags' unique `id` attribute is shown in the drop-down field. Individual tags can be selected from the list.
  - With the help of the "Set tag position" button, the position of individual tags can be set manually. The position of the tag currently selected in the drop-down field is set on the next mouse click in the layout view.

- *NFC*: The NFC panel works analogously to the Ubisense panel with the difference that NFC positions cannot be set manually because the location of NFC tags is considered to be static data. The "Simulate tag contact" button sets the timestamp of the selected tag to the current time.

- *Reasoner*: In addition to dynamically starting and stopping the reasoner, which is useful to demonstrate the effect of registered rules, the available geometric zones can be visualized. Selecting a zone from the list and clicking the "Show / Hide selected zone" button causes the layout view to show the zone.

- *Snapshot*: With the help of the "Save snapshot" button it is possible to save snapshots of the context model to any folder as an XML document. The "Change snapshot directory" button can be used to change this folder. The filename is automatically determined by the current time.

All of the values and options update if the positions change or new objects are added to the model. The GUI can therefore be used effectively for testing and simulation.



UbiSense reader is running and fetching data from http://192.168.1.210:49185/

020-000-016-175

NFC reader is not running.

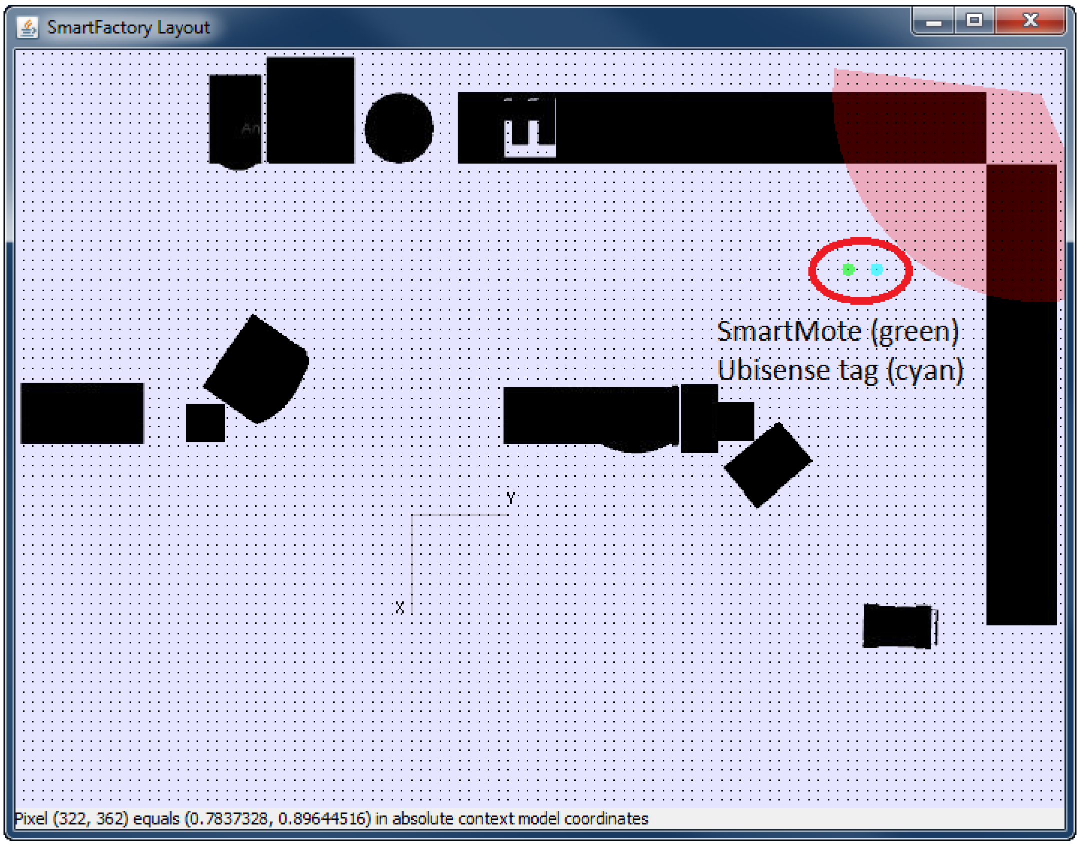
ctx_obj_2_1

Reasoner is running.

zone_smartfactory_gesamt

Ready to save snapshot to context/snapshots/

Figure 7.7.: Control GUI with options to control the individual components



Pixel (322, 362) equals (0.7837328, 0.89644516) in absolute context model coordinates

Figure 7.8.: Factory overview. The positions of the Ubisense tags and the SmartMote are continually updated.

8. Feasibility Demonstration

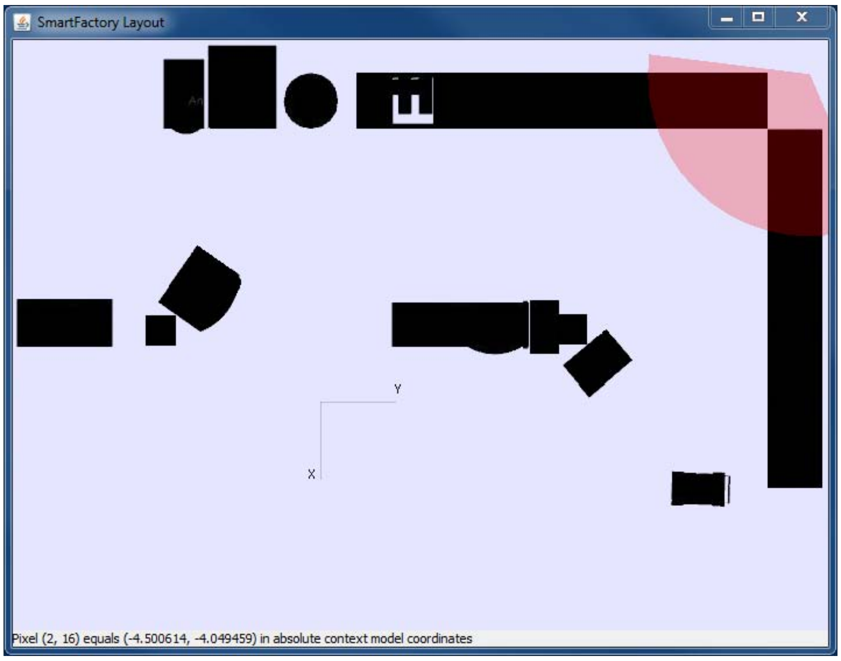
To prove the feasibility of the modified SmartMote and test the system under real-world conditions an experiment in the *SmartFactory*$^{\mathbf{KL}}$ was conducted. The goal was to see if

- the sensors behave according to the specification.
- the Ubisense system can provide reliable position data or if there are any "blind spots" where the statistical correction cannot compensate for inaccurate sensor data.
- the zone types are correctly implemented and offer location-dependent behavior.
- the GUI reacts fast enough to zone changes or if there are delays during the update process.

To test the sensors, the SmartMote was carried to various locations in the factory. The sensed Ubisense location as well as the derived SmartMote position was observed on the control GUI. For the third point a context model containing different zones for the "mixing" module was created. The RUM's device compound entry for this module was extended to include two interaction zones pointing to the zones in the context model and one default zone. The zone nearest to the mixing module was defined as an arc and is shown in Figure 8.1(a). For this zone all available functionality was allowed. The second zone (cf. Figure 8.1(b)) comprised most of the modules and the majority of the factory hall in a block. The task model for this zone allowed only to read parameters, but not to set them. The default zone contained an empty use model. The figures on the next pages show the user's view, the control GUI and the generated SmartMote GUI in different situations.

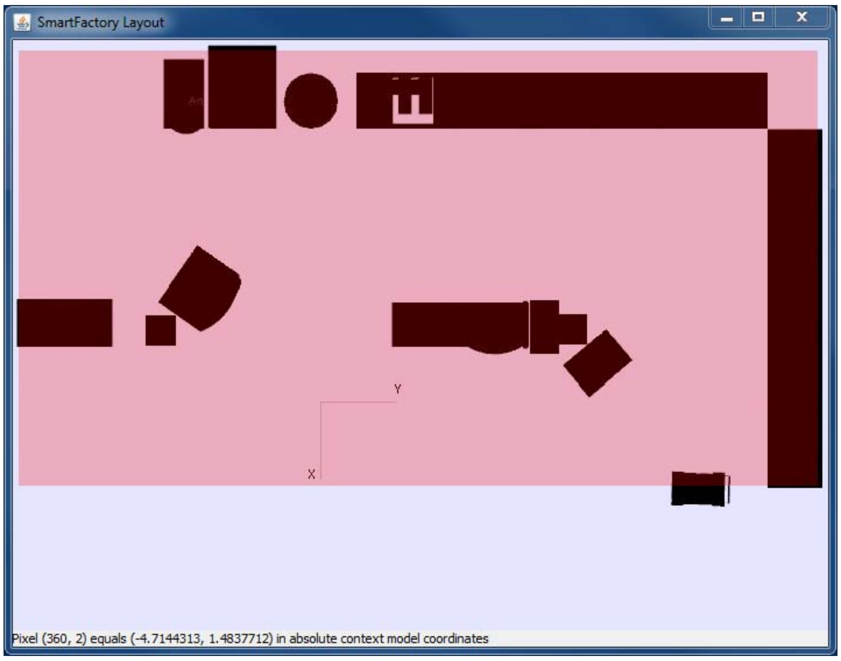
Unfortunately, it became clear during simulations that the new GUI version with included usability patterns was not able to handle updates in the task model efficiently. It took up to three seconds until the GUI adapted and even then it "forgot" the activated device compound and the user had to select it again from the navigation bar. The test was therefore performed with the original version of the GUI [Mas08] which performed very well.

The test showed that the Ubisense location information can be considered accurate. While there were some areas in which the location tended to deviate from the real position, the correction could always produce a sufficiently accurate position. The value for the statistical smoothing was reduced from the initial 10 to 3. The rest of the test also progressed without errors. The user could wander through the factory and the GUI adapted according to the defined zones with a delay of about two seconds at maximum. The integration of the context engine into the SmartMote can therefore be rated as a success.



Pixel (2, 16) equals (-4.500614, -4.049459) in absolute context model coordinates

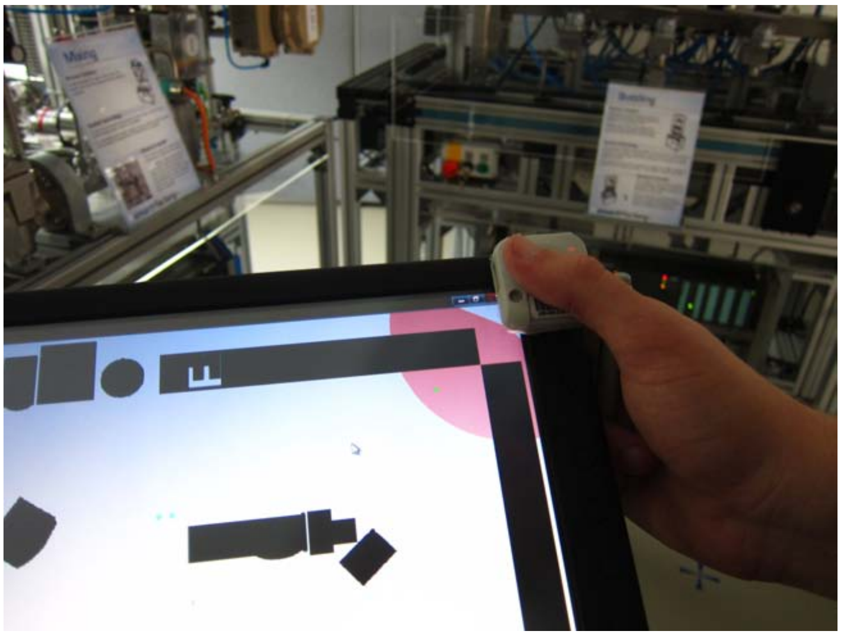
(a) Nearest zone, defined as arc



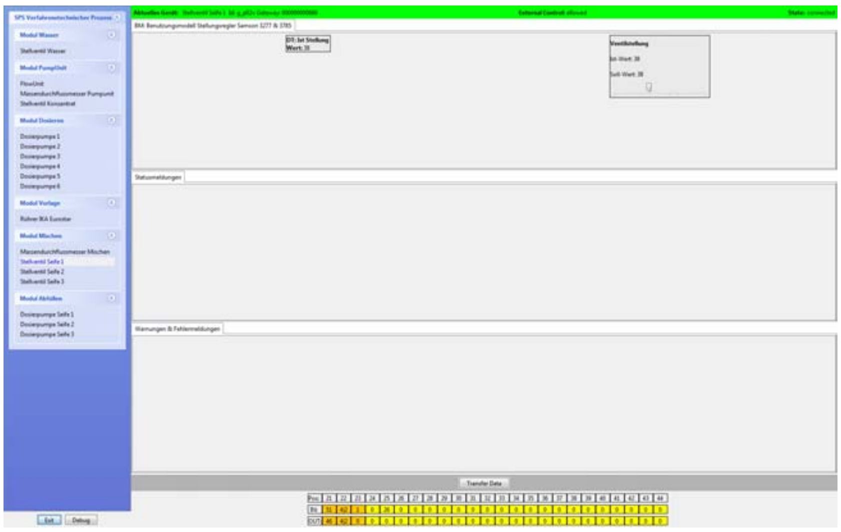
Pixel (360, 2) equals (-4.7144313, 1.4837712) in absolute context model coordinates

(b) Middle zone comprising most of the modules

Figure 8.1.: Three zones were used in the test. The third zone is the default and defined implicitly. Any location not included in the first two zones is part of the default zone.



(a) The layout GUI on the SmartMote and a view on the mixing module in the background



(b) The corresponding generated GUI. The module offers full functionality in this zone.

Figure 8.2.: User and SmartMote are in the nearest zone.

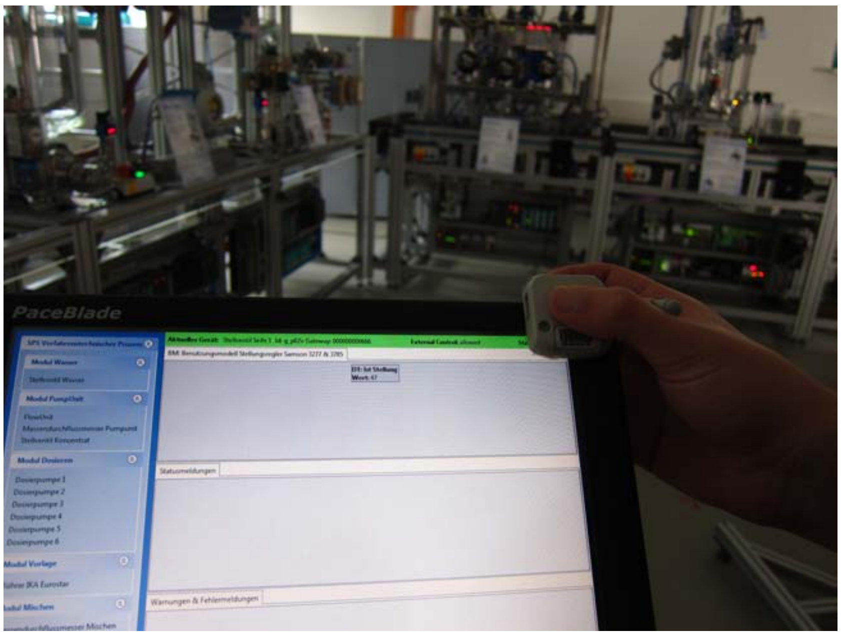


Figure 8.3.: The generated GUI shows read-only parameters. Setting parameters is disabled from the second zone.

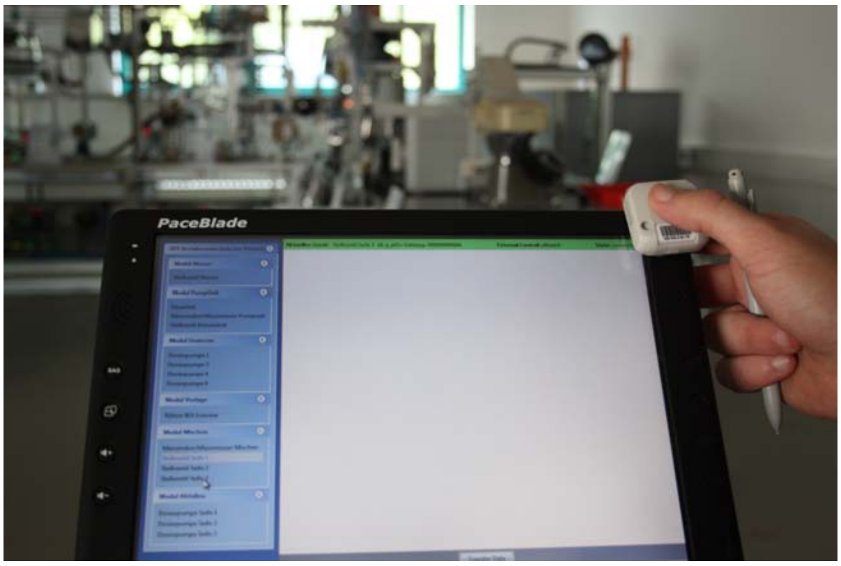


Figure 8.4.: No interaction at all is allowed from the default zone.

9. Conclusion and Future Work

Conclusion

The goal of this work was to develop an infrastructure for the context-based adaptation of the SmartMote. After introducing the *SmartFactory*$^{\mathbf{KL}}$, its universal control device (UCD), the SmartMote, and considering the special properties of ambient intelligence the notion of context and context-awareness was formally introduced. Examples of context-aware devices were given to show the wide area of possible applications. The classifications discussed led to the observation that some context types are more important than others and that low-level context from sensors can be refined to high-level information about the situation. Existing approaches for separating context from the application logic in a context model were examined. Especially the advantages and shortcomings of different modeling techniques and the properties of different location models were inspected.

The primary contribution of this thesis is the development of a formal context model and a run-time architecture for reasoning on this model to support context-adaptation in the SmartMote's model-based user interface development process (MBUID). Using the *SmartFactory*$^{\mathbf{KL}}$ as an example of ambient intelligent production environments, the model considers the location of various objects and the geometric extents of three types of zones. Through the provided access mechanisms, two sensor technologies are connected to the model. The Ubisense and NFC components continuously update the model with up-to-date location information. As the model is based on XML, the specialized XMLBeans tool can be used to bind the XML schema to Java types. This combines the interoperability of markup languages with the performance of object-oriented modeling. The XMLBeans compiler is then extended with a notification mechanism that allows clients to react to changes in the model. Future additions to the model can be made by extending the XML schema without affecting the existing components.

The developed reasoner component is based on the context model. With an internal set of rules that operate on the model it is possible to derive high-level context data from current and previous sensor input. Four rules were defined, two to keep the model from becoming too overloaded with unnecessary information and two to enrich it with high-level context information. Combining the available information it is possible to determine a list of zones the user is currently in. Rules are based on a simple interface so that it is easy for developers to create and add their own rules. Rules can be added and removed during run-time. Overall, the system provides a clear and extensible architecture for integrating context into the MBUID process.

The feasibility of the approach has been demonstrated using the concept of interaction zones. The Room-based Use Model (RUM) allows the specification of different task models depending on the zones the user is currently in. To support this scenario, the reasoner also mediates between the previous MBUID process and the context model. Using the zone information from the context model the reasoner transforms the RUM from its context-sensitive form (including all zone definitions) to a context-insensitive form corresponding to the current context of use (including only the task models of active zones). For the first time it is possible to adapt the SmartMote GUI not only to the set of available devices, but also to the location of the user.

The system has been successfully tested in the *SmartFactory*$^{\mathbf{KL}}$, which means scenarios like the one given in the introduction are possible. The reduction of complexity in ambient intelligent production environments has therefore been further improved. It is now not only possible to control all devices from a single mobile device, but also to exclude tasks that are not viable from the user's current location. These improvements have taken the SmartMote one step further towards the vision of a human-centered ambient intelligent production environment.

Future Work

While the feasibility of the engine has been shown and the system has been tested in the *SmartFactory*$^{\mathbf{KL}}$, the expected positive impact on usability and safety can be evaluated in future studies.

From an conceptual view, the RUM provides chances for improvement. The monolithic approach, in which all context-sensitive information is included in one model, leads to a hardly manageable model. Especially the creation of such a complex model becomes harder and more error-prone if many situations are considered. A separation into different models could facilitate their construction.

Some obstacles with XMLBeans also arose. While XMLBeans provides a reliable transformation from XML to Java, the resulting object tree contains no backward references. Accessing a parent node is only possible by switching to an XML representation or by traversing the tree again from the root. Future improvements will also be made to support the context engine together with the pattern-based MBUID process.

As there is now an infrastructure for the handling of context available, additional context-dependent behavior can be integrated. Developing more location-based functionality as well as considering more types of context like noise levels, user preferences or device profiles could be the goal of future projects. Providing context-sensitive support of users' workflows by predicting their next steps and tasks that they need to perform next is also a possibility.

Finally, the restriction of exclusive access to the resources could be lifted. The SmartMote currently assumes that only one universal control device has access to the factory. Allowing multiple UCDs opens up new topics of interest. Tasks from a local task model could be performed in cooperation with other users. The distribution of tasks to multiple devices raises new questions regarding the intelligent sharing of contextual data as well.

A. Appendix

A.1. Context Model: XML Schema

```
<?xml version="1.0" encoding="UTF-8"?>
<schema targetNamespace="generated.model.context.gabigui.agse"
     elementFormDefault="qualified" xmlns="http://www.w3.org/2001/XMLSchema"
     xmlns:smctx="generated.model.context.gabigui.agse">

    <element name="Context" type="smctx:TContext"></element>

    <complexType name="TContext">
      <sequence maxOccurs="1" minOccurs="1">
        <element name="Objects" type="smctx:TObjects" maxOccurs="1" minOccurs="1"></element>
        <element name="Places" type="smctx:TPlaces" maxOccurs="1" minOccurs="1"></element>
      </sequence>
    </complexType>

    <complexType name="TObjects">
      <sequence maxOccurs="1" minOccurs="1">
        <element name="SmartMote" type="smctx:TSmartMote" maxOccurs="1" minOccurs="1"></element>
        <element name="NFCTag" type="smctx:TNFCTag" maxOccurs="unbounded" minOccurs="0"></element>
        <element name="UbisenseTag" type="smctx:TUbisenseTag" maxOccurs="unbounded" minOccurs="0"></element>
      </sequence>
    </complexType>

    <complexType name="TSmartMote">
      <sequence maxOccurs="1" minOccurs="1">
        <element name="PositionHistory"
          type="smctx:TPositionHistory" maxOccurs="1" minOccurs="1">
        </element>
      </sequence>
      <attribute name="id" type="ID" use="required"></attribute>
      <attribute name="ubisenseTagID" type="IDREF"></attribute>
    </complexType>

    <complexType name="TPositionHistory">
      <sequence maxOccurs="1" minOccurs="1">
        <element name="PointReference" maxOccurs="unbounded" minOccurs="0" type="smctx:TPointReference"></element>
      </sequence>
      <attribute name="maxItems">
        <simpleType>
          <restriction base="int">
            <minExclusive value="2"></minExclusive>
          </restriction>
        </simpleType>
      </attribute>
    </complexType>

    <simpleType name="TType">
      <restriction base="string">
        <enumeration value="static"></enumeration>
        <enumeration value="sensed"></enumeration>
        <enumeration value="derived"></enumeration>
        <enumeration value="dynamic"></enumeration>
        <enumeration value="profiled"></enumeration>
      </restriction>
    </simpleType>


    <complexType name="TNFCTag">
      <sequence maxOccurs="1" minOccurs="1">
        <element name="PositionHistory"
          type="smctx:TPositionHistory" maxOccurs="1" minOccurs="1">
        </element>
        <element name="Firma" type="string" maxOccurs="1" minOccurs="1"></
             element>
        <element name="Geraetetyp" type="string" maxOccurs="1" minOccurs="1"></
             element>
        <element name="Geraetetyp_Id" type="string" maxOccurs="1" minOccurs="1"
             ></element>
        <element name="Geraetenummer" type="string" maxOccurs="1" minOccurs="1"
             ></element>
        <element name="Geraete_Id" type="string" maxOccurs="1" minOccurs="1"></
             element>
        <element name="Geraete_Name" type="string" maxOccurs="1" minOccurs="1">
             </element>
        <element name="Modul_Name" type="string" maxOccurs="1" minOccurs="1"></
             element>
        <element name="Modul_Nummer" type="string" maxOccurs="1" minOccurs="1">
             </element>
      </sequence>
      <attribute name="id" type="ID" use="required"></attribute>
    </complexType>

    <complexType name="TUbisenseTag">
      <sequence maxOccurs="1" minOccurs="1">
        <element name="Name" type="string" maxOccurs="1" minOccurs="1"></
             element>
        <element name="PositionHistory" type="smctx:TPositionHistory" maxOccurs
             ="1" minOccurs="1"></element>
      </sequence>
      <attribute name="id" type="ID" use="required"></attribute>
    </complexType>

    <complexType name="TPlaces">
      <sequence maxOccurs="1" minOccurs="1">
        <element name="Coordinates" type="smctx:TCoordinates" maxOccurs="1"
             minOccurs="1"></element>
        <element name="Positions" type="smctx:TPositions" maxOccurs="1"
             minOccurs="1"></element>
      </sequence>
    </complexType>

    <complexType name="TCoordinates">
      <sequence maxOccurs="1" minOccurs="1">
        <element name="Block" type="smctx:TBlock" maxOccurs="unbounded"
             minOccurs="0"></element>
        <element name="Ring" type="smctx:TRing" maxOccurs="unbounded" minOccurs
             ="0"></element>
        <element name="CircleSegment" type="smctx:TCircleSegment" maxOccurs="
             unbounded" minOccurs="0"></element>
      </sequence>
    </complexType>

    <complexType name="TBlock">
      <sequence maxOccurs="1" minOccurs="1">
        <element name="SourceVertex" type="smctx:TPointReference"
          maxOccurs="1" minOccurs="1">
        </element>
        <element name="DestinationVertex"
```

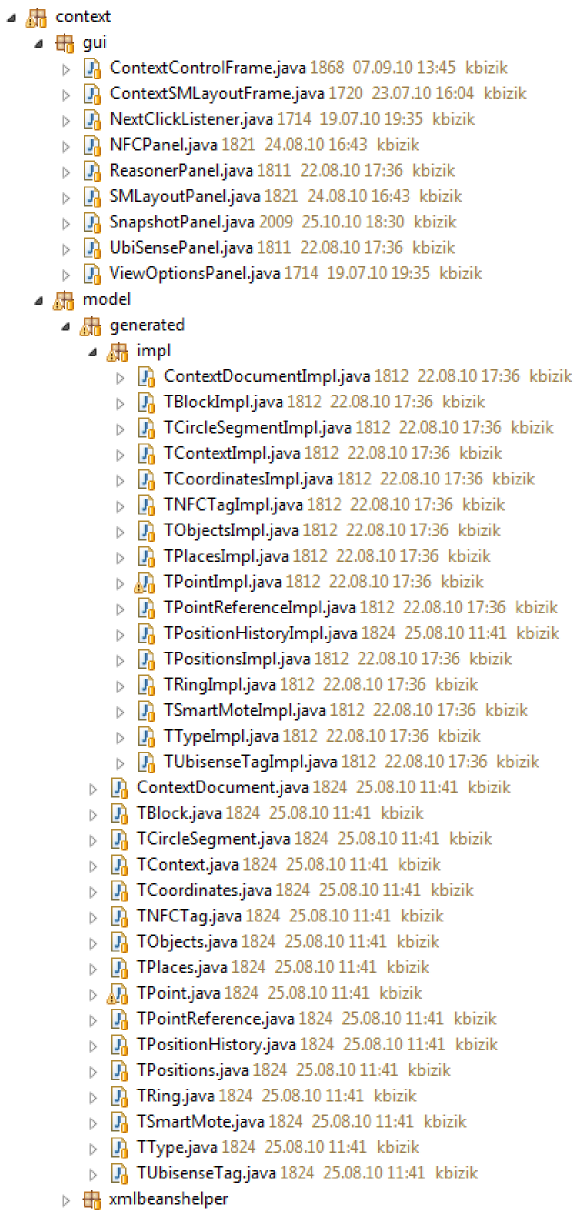
```
          type="smctx:TPointReference" maxOccurs="1" minOccurs="1">
        </element>
        <element name="Height" type="double" maxOccurs="1" minOccurs="1"></
             element>
      </sequence>
      <attribute name="id" type="ID" use="required"></attribute>
    </complexType>
    <complexType name="TPointReference">
      <attribute name="time" type="dateTime"></attribute>
      <attribute name="pointId" type="IDREF"></attribute>
    </complexType>

    <complexType name="TRing">
      <sequence maxOccurs="1" minOccurs="1">
        <element name="CenterBottomVertex"
          type="smctx:TPointReference" maxOccurs="1" minOccurs="1">
        </element>
        <element name="InnerRadius" type="double" maxOccurs="1" minOccurs="1"><
             /element>
        <element name="OutterRadius" type="double" maxOccurs="1" minOccurs="1">
             </element>
        <element name="Height" type="double" maxOccurs="1" minOccurs="1"></
             element>
      </sequence>
      <attribute name="id" type="ID" use="required"></attribute>
    </complexType>

    <complexType name="TCircleSegment">
      <sequence maxOccurs="1" minOccurs="1">
        <element name="CenterBottomVertex"
          type="smctx:TPointReference" maxOccurs="1" minOccurs="1">
        </element>
        <element name="OutterBottomVertex"
          type="smctx:TPointReference" maxOccurs="1" minOccurs="1">
        </element>
        <element name="Height" type="double" maxOccurs="1" minOccurs="1"></
             element>
        <element name="Angle" type="int" maxOccurs="1"
          minOccurs="1">
        </element>
      </sequence>
      <attribute name="id" type="ID" use="required"></attribute>
    </complexType>

    <complexType name="TPositions">
      <sequence maxOccurs="1" minOccurs="1">
        <element name="Point" type="smctx:TPoint" maxOccurs="unbounded"
             minOccurs="0"></element>
      </sequence>
    </complexType>

    <complexType name="TPoint">
      <sequence maxOccurs="1" minOccurs="1">
        <element name="X" type="double" maxOccurs="1"
          minOccurs="1">
        </element>
        <element name="Y" type="double" maxOccurs="1"
          minOccurs="1">
        </element>
        <element name="Z" type="double" maxOccurs="1"
          minOccurs="1">
        </element>
      </sequence>
```

```
            <attribute name="id" type="ID" use="required"></attribute>
            <attribute name="containedInCoordinates" type="IDREFS"></attribute>
        </complexType>
</schema>
```

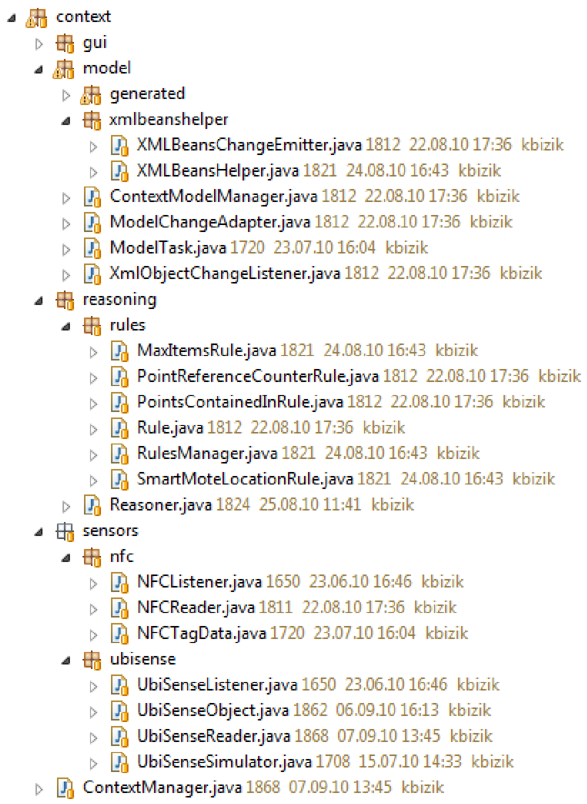
A.2. Example Ubisense XML File

```xml
<ArrayOfUbisenseObject>
  <UbisenseObject>
    <Id>Id1</Id>
    <Name>Name1</Name>
    <X>39.307106</X>
    <Y>3.977844</Y>
    <Z>1.3933041</Z>
    <LastSeenOn>0001-01-01T00:00:00</LastSeenOn>
  </UbisenseObject>
  <UbisenseObject>
    <Id>Id2</Id>
    <Name>Name2</Name>
    <X>8.380611</X>
    <Y>38.08979</Y>
    <Z>1.0</Z>
    <LastSeenOn>0001-01-01T00:00:00</LastSeenOn>
  </UbisenseObject>
    <UbisenseObject>
    <Id>Id3</Id>
    <Name>Name3</Name>
    <X>33.177</X>
    <Y>47.199135</Y>
    <Z>1.8</Z>
    <LastSeenOn>0001-01-01T00:00:00</LastSeenOn>
  </UbisenseObject>
</ArrayOfUbisenseObject>
```

A.3. Complete List of Packages and Classes



```
◢ context
  ◢ gui
    ▷ ContextControlFrame.java 1868  07.09.10 13:45  kbizik
    ▷ ContextSMLayoutFrame.java 1720  23.07.10 16:04  kbizik
    ▷ NextClickListener.java 1714  19.07.10 19:35  kbizik
    ▷ NFCPanel.java 1821  24.08.10 16:43  kbizik
    ▷ ReasonerPanel.java 1811  22.08.10 17:36  kbizik
    ▷ SMLayoutPanel.java 1821  24.08.10 16:43  kbizik
    ▷ SnapshotPanel.java 2009  25.10.10 18:30  kbizik
    ▷ UbiSensePanel.java 1811  22.08.10 17:36  kbizik
    ▷ ViewOptionsPanel.java 1714  19.07.10 19:35  kbizik
  ◢ model
    ◢ generated
      ◢ impl
        ▷ ContextDocumentImpl.java 1812  22.08.10 17:36  kbizik
        ▷ TBlockImpl.java 1812  22.08.10 17:36  kbizik
        ▷ TCircleSegmentImpl.java 1812  22.08.10 17:36  kbizik
        ▷ TContextImpl.java 1812  22.08.10 17:36  kbizik
        ▷ TCoordinatesImpl.java 1812  22.08.10 17:36  kbizik
        ▷ TNFCTagImpl.java 1812  22.08.10 17:36  kbizik
        ▷ TObjectsImpl.java 1812  22.08.10 17:36  kbizik
        ▷ TPlacesImpl.java 1812  22.08.10 17:36  kbizik
        ▷ TPointImpl.java 1812  22.08.10 17:36  kbizik
        ▷ TPointReferenceImpl.java 1812  22.08.10 17:36  kbizik
        ▷ TPositionHistoryImpl.java 1824  25.08.10 11:41  kbizik
        ▷ TPositionsImpl.java 1812  22.08.10 17:36  kbizik
        ▷ TRingImpl.java 1812  22.08.10 17:36  kbizik
        ▷ TSmartMoteImpl.java 1812  22.08.10 17:36  kbizik
        ▷ TTypeImpl.java 1812  22.08.10 17:36  kbizik
        ▷ TUbisenseTagImpl.java 1812  22.08.10 17:36  kbizik
      ▷ ContextDocument.java 1824  25.08.10 11:41  kbizik
      ▷ TBlock.java 1824  25.08.10 11:41  kbizik
      ▷ TCircleSegment.java 1824  25.08.10 11:41  kbizik
      ▷ TContext.java 1824  25.08.10 11:41  kbizik
      ▷ TCoordinates.java 1824  25.08.10 11:41  kbizik
      ▷ TNFCTag.java 1824  25.08.10 11:41  kbizik
      ▷ TObjects.java 1824  25.08.10 11:41  kbizik
      ▷ TPlaces.java 1824  25.08.10 11:41  kbizik
      ▷ TPoint.java 1824  25.08.10 11:41  kbizik
      ▷ TPointReference.java 1824  25.08.10 11:41  kbizik
      ▷ TPositionHistory.java 1824  25.08.10 11:41  kbizik
      ▷ TPositions.java 1824  25.08.10 11:41  kbizik
      ▷ TRing.java 1824  25.08.10 11:41  kbizik
      ▷ TSmartMote.java 1824  25.08.10 11:41  kbizik
      ▷ TType.java 1824  25.08.10 11:41  kbizik
      ▷ TUbisenseTag.java 1824  25.08.10 11:41  kbizik
    ▷ xmlbeanshelper
```



```
◢ context
  ▷ gui
  ◢ model
    ▷ generated
    ◢ xmlbeanshelper
      ▷ XMLBeansChangeEmitter.java 1812  22.08.10 17:36  kbizik
      ▷ XMLBeansHelper.java 1821  24.08.10 16:43  kbizik
    ▷ ContextModelManager.java 1812  22.08.10 17:36  kbizik
    ▷ ModelChangeAdapter.java 1812  22.08.10 17:36  kbizik
    ▷ ModelTask.java 1720  23.07.10 16:04  kbizik
    ▷ XmlObjectChangeListener.java 1812  22.08.10 17:36  kbizik
  ◢ reasoning
    ◢ rules
      ▷ MaxItemsRule.java 1821  24.08.10 16:43  kbizik
      ▷ PointReferenceCounterRule.java 1812  22.08.10 17:36  kbizik
      ▷ PointsContainedInRule.java 1812  22.08.10 17:36  kbizik
      ▷ Rule.java 1812  22.08.10 17:36  kbizik
      ▷ RulesManager.java 1821  24.08.10 16:43  kbizik
      ▷ SmartMoteLocationRule.java 1821  24.08.10 16:43  kbizik
    ▷ Reasoner.java 1824  25.08.10 11:41  kbizik
  ◢ sensors
    ◢ nfc
      ▷ NFCListener.java 1650  23.06.10 16:46  kbizik
      ▷ NFCReader.java 1811  22.08.10 17:36  kbizik
      ▷ NFCTagData.java 1720  23.07.10 16:04  kbizik
    ◢ ubisense
      ▷ UbiSenseListener.java 1650  23.06.10 16:46  kbizik
      ▷ UbiSenseObject.java 1862  06.09.10 16:13  kbizik
      ▷ UbiSenseReader.java 1868  07.09.10 13:45  kbizik
      ▷ UbiSenseSimulator.java 1708  15.07.10 14:33  kbizik
  ▷ ContextManager.java 1868  07.09.10 13:45  kbizik
```

List of Figures

List of Tables

Listings

Bibliography

[ADB$^+$99] Gregory Abowd, Anind Dey, Peter Brown, Nigel Davies, Mark Smith, and Pete Steggles. Towards a Better Understanding of Context and Context-Awareness. In Hans-W Gellersen, editor, *Handheld and Ubiquitous Computing*, volume 1707 of *Lecture notes in computer science*, pages 304–307. Springer Berlin / Heidelberg, 1999.

[Adv22] Advanced Card Systems Ltd. ACR122U NFC Contactless Smart Card Reader, Last Checked: 2010/10/22. `http://www.acs.com.hk/index.php?pid=product&id=ACR122U`.

[AM03] Emile Aarts and Stefano Marzano. *The new everyday, Views on ambient intelligence.* 010 Publ., Rotterdam, 2003.

[Ame13] American Telephone and Telegraph Corporation. Laboratories Cambridge Archives - The Active Badge, Last checked: 2010/10/13. `http://www.cl.cam.ac.uk/research/dtg/attarchive/thebadge.html`.

[Apa22] Apache XMLBeans. XMLBeans Project Homepage, Last checked: 2010/10/22. `http://xmlbeans.apache.org/`.

[Arn06] Henrik Arndt. *Integrierte Informationsarchitektur, Die erfolgreiche Konzeption professioneller Websites.* X.media.press. Springer and Springer-Verlag Berlin Heidelberg, Berlin, Heidelberg, 2006.

[AVSC06] Mark Altosaar, Roel Vertegaal, Changuk Sohn, and Daniel Cheng. AuraOrb: using social awareness cues in the design of progressive notification appliances. In *Proceedings of the 18th Australia conference on Computer-Human Interaction: Design: Activities, Artefacts and Environments*, pages 159–166. ACM, Sydney, Australia, 2006.

[Ay07] Feruzan Ay. Context Modeling and Reasoning using Ontologies. Berlin, 2007.

[Bar05] Jakob E. Bardram. The Java Context Awareness Framework (JCAF) – A Service Infrastructure and Programming Framework for Context-Aware Applications. In Hans W. Gellersen, Albrecht Schmidt, and Roy Want, editors, *Pervasive Computing (vol. 3468)*, volume 3468 of *Lecture notes in computer science*, pages 98–115. Springer-Verlag GmbH, Berlin Heidelberg, 2005.

[BAT98] Jason A. Brotherton, Gregory D. Abowd, and Khai N. Truong. Supporting capture and access interfaces for informal and opportunistic meetings, Georgia Tech Technical Report GIT-GVU-99-06, 1998.

[Bau03] Joseph Bauer. *Identification and Modeling of Contexts for Different Information Scenarios in Air Traffic.* Diploma Thesis, Technische Universität Berlin, Berlin, 2003.

[BBC97] P. J. Brown, J. D. Bovey, and Xian Chen. Context-aware applications: from the laboratory to the marketplace, Personal Communications, IEEE. *Personal Communications, IEEE DOI - 10.1109/98.626984*, 4(5):58–64, 1997.

[BBH97] J. Bacon, J. Bates, and D. Halls. Location-Oriented Multimedia. *Personal Communications, IEEE*, 4(5):48–57, 1997.

[BCQ$^+$07] Cristiana Bolchini, Carlo A. Curino, Elisa Quintarelli, Fabio A. Schreiber, and Letizia Tanca. A Data-Oriented Survey of Context Models. *ACM SIGMOD Record*, 36(4):19–26, 2007.

[BD05] Christian Becker and Frank Dürr. On location models for ubiquitous computing. *Personal and Ubiquitous Computing*, 9(1):20–31, 2005.

[BDR07] Matthias Baldauf, Schahram Dustdar, and Florian Rosenberg. A survey on context-aware systems. *Int. J. Ad Hoc Ubiquitous Comput.*, 2(4):263–277, 2007.

[BGM$^+$09] Kai Breiner, Daniel Görlich, Oliver Maschino, Gerrit Meixner, and Detlef Zühlke. Run-Time Adaptation of a Universal User Interface for Ambient Intelligent Production Environments. In Julie A. Jacko, editor, *Human-computer interaction*, volume 5613 of *Lecture notes in computer science*, pages 663–672. Springer, Berlin, 2009.

[BGSG10] Kai Breiner, Volkmar Gauckler, Marc Seissler, and Gerrit Meixner. Evaluation of user interface adaptation strategies in the process of model-driven user interface development. In *Proceedings of the 5th International Workshop on Model-Driven Development of Advanced User Interfaces. International Workshop on Model Driven Development of Advanced User Interfaces (MDDAUI-2010), located at CHI 2010, April 10, Atlanta,, GA, United States.* CEUR-Proceedings, 2010.

[BLFM23] T. Berners-Lee, R. Fielding, and L. Masinter. RFC 3986 - Uniform Resource Identifier (URI): Generic Syntax, Last checked: 2010/10/23. `http://tools.ietf.org/html/rfc3986`.

[BMGM09] Kai Breiner, Oliver Maschino, Daniel Görlich, and Gerrit Meixner. Towards automatically interfacing application services integrated in a automated model based user interface generation process. In Gerrit Meixner, Daniel Görlich, K. Breiner, H. Hußmann, A. Pleuß, S. Sauer, and J. Van den Bergh, editors, *4th International Workshop on Model Driven Development of Advanced User Interfaces. International Workshop on Model Driven Development of Advanced User Interfaces (MDDAUI-2009), February 8, Sanibel Island,, Florida, United States*, volume 439 of *CEUR Workshop Proceedings, ISSN 1613-0073.* CEUR Workshop Proceedings (Online), 2009.

[BRJ05] Grady Booch, James Rumbaugh, and Ivar Jacobson. *The Unified Modeling Language User Guide (UML), The Ultimate Tutorial from the original Designers - Covers UML 2.0.* Addison Wesley, 2005.

[Bro96] P. J. Brown. The Stick-e Document: a Framework for Creating Context-aware Applications. In *Proceedings of EP'96, Palo Alto*, pages 259–272, 1996.

[Bro98] P. Brown. Triggering information by context. *Personal and Ubiquitous Computing*, 2(1):18–27, 1998.

[CCT+03] Gaëlle Calvary, Joëlle Coutaz, David Thevenin, Quentin Limbourg, Laurent Bouillon, and Jean Vanderdonckt. A Unifying Reference Framework for multi-target user interfaces, Computer-Aided Design of User Interface. *Interacting with Computers*, 15(3):289–308, 2003.

[CF03] Ekaterina Chtcherbina and Marquart Franz. Peer-to-peer coordination framework (p2pc): Enabler of mobile ad-hoc networking for medicine, business, and entertainment. In *Proceedings of International Conference on Advances in Infrastructure for Electronic Business, Education, Science, Medicine, and Mobile Technologies on the Internet*, 2003.

[CFJ03] Harry Chen, Tim Finin, and Anupam Joshi. An Intelligent Broker for Context-Aware Systems. In *IN ADJUNCT PROCEEDINGS OF UBICOMP*, 2003.

[CK00] Guanling Chen and David Kotz. A survey of context-aware mobile computing research, Dartmouth Computer Science Technical Report TR2000-381, 2000.

[CLC04] Tim Clerckx, Kris Luyten, and Karin Coninx. The mapping problem back and forth: customizing dynamic models while preserving consistency. In *TAMODIA '04: Proceedings of the 3rd annual conference on Task models and diagrams*, pages 33–42, New York, NY, USA, 2004. ACM.

[CLC05] Tim Clerckx, Kris Luyten, and Karin Coninx. Generating Context-Sensitive Multiple Device Interfaces from Design. In Robert Jacob, Quentin Limbourg, and Jean Vanderdonckt, editors, *Computer-Aided Design of User Interfaces IV*, pages 283–296. Springer Netherlands, 2005.

[CPFJ04] Harry Chen, Filip Perich, Tim Finin, and Anupam Joshi. SOUPA: Standard Ontology for Ubiquitous and Pervasive Applications. In Imrich Chlamtac, editor, *Proceedings of MobiQuitous 2004*, pages 258–267. IEEE Service Center, Piscataway, NJ, 2004.

[Deu16] Deutsches Forschungszentrum für Künstliche Intelligenz GmbH. German Research Center for Artificial Intelligence, Last checked: 2010/10/16. `http://www.dfki.de/`.

[Dij20] Edsger W. Dijkstra. E.W. Dijkstra Archive: On the role of scientific thought (EWD447), Last checked: 2010/10/20. `http://www.cs.utexas.edu/users/EWD/transcriptions/EWD04xx/EWD447.html`.

[DKM18] Martin Dr. Kempa and Zoltán Ádám Mann. Model Driven Architecture - Gesellschaft für Informatik e.V., Last checked: 2010/10/18. `http://www.gi-ev.de/no_cache/service/informatiklexikon/informatiklexikon-detailansicht/meldung/model-driven-architecture-117`.

[DR03] Frank Dürr and Kurt Rothermel. On a Location Model for Fine-Grained Geocast. In *UbiComp 2003: Ubiquitous Computing*, volume 2864 of *Lecture notes in computer science*, pages 18–35. Springer Berlin / Heidelberg, 2003.

[Ecl22] Eclipse Foundation Inc. Eclipse IDE Homepage, Last checked: 2010/10/22. `http://eclipse.org/`.

[EHL01] M. Ebling, G. D. H. Hunt, and H. Lei. Issues for context services for pervasive computing, Middleware 2001 Workshop on Middleware for Mobile Computing, Heidelberg, 2001.

[Esp08] Esprit project 26900. Technology for Enabling Awareness, Last checked: 2010/09/08. `http://www.teco.edu/tea/`.

[FHM+08] Alois Ferscha, Manfred Hechinger, Rene Mayrhofer, Ekaterina Chtcherbina, Marquart Franz, Marcos dos Santos Rocha, and Andreas Zeidler. Bridging the Gap with P2P Patterns, 2008.

[Fra14] Fraunhofer Gesellschaft. Integrated Publication and Information Systems Institute - Material Repository, Last checked: 2010/10/14. `http://www.ipsi.fraunhofer.de/ambiente/material/pictures/`.

[GÖ9] Daniel Görlich. *Laufzeit-Adaption von Benutzungsschnittstellen für Ambient-Intelligence-Umgebungen mittels raumbasierter Benutzungsmodelle, PhD-Thesis*, volume 20 of *Fortschritt-Berichte pak Mensch-Maschine-Interaktion.* Techn. Univ., Kaiserslautern, 2009.

[GHJV09] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design patterns, Elements of reusable object-oriented software.* Safari Tech Books Online. Addison-Wesley, Boston, 37th print. edition, 2009.

[HI04] K. Henricksen and J. Indulska. Modelling and using imperfect context information, Proceedings of the Second IEEE Annual Conference on Pervasive Computing and Communications Workshops, 2004.

[HIMB05] Karen Henricksen, Jadwiga Indulska, Ted McFadden, and Sasitharan Balasubramaniam. Middleware for Distributed Context-Aware Systems. In Robert Meersman and Zahir Tari, editors, *On the Move to Meaningful Internet Systems 2005: CoopIS, DOA, and ODBASE*, volume 3760 of *Lecture notes in computer science*, pages 846–863. Springer Berlin / Heidelberg, 2005.

[HIR03] Karen Henricksen, Jadwiga Indulska, and Andry Rakotonirainy. Generating Context Management Infrastructure from High-Level Context Models. In *4th International Conference on Mobile Data Management (MDM) - Industrial Track, 21-24 January 2003, Melbourne, Australia*, pages 1–6, 2003.

[HMHW08] Volker Haarslev, Ralf Möller, Kay Hidde, and Michael Wessel. Renamed Abox and Concept Expression Reasoner, Last checked: 2010/09/08. `http://www.sts.tu-harburg.de/~r.f.moeller/racer/`.

[HWM+03] Thomas Hofer, Wieland Schwinger, Mario Pichler, Gerhard Leonhartsberger, Josef Altmann, and Werner Retschitzegger. Context-Awareness on Mobile Devices - the Hydrogen Approach. In Ralph H. Sprague, editor, *Proceedings of the 36th Annual Hawaii International Conference on System Sciences*, volume 9, pages 292a–292a. IEEE Computer Soc., Los Alamitos, Calif., 2003.

[IRRH03] Jadwiga Indulska, Ricky Robinson, Andry Rakotonirainy, and Karen Henricksen. Experiences in Using CC/PP in Context-Aware Systems. In Ming-Syan Chen, Panos Chrysanthis, Morris Sloman, and Arkady Zaslavsky, editors, *Mobile Data Management*, volume 2574 of *Lecture notes in computer science*, pages 247–261. Springer Berlin, Heidelberg, 2003.

[Jav22] Java-Homepage, Last checked: 2010/10/22. `http://www.oracle.com/technetwork/java/index.html`.

[Jen08] Jena. A Semantic Web Framework for Java, Last checked: 2010/09/08. `http://jena.sourceforge.net/`.

[Kha09] Nadeem Khan. Identifikation und automatisierte Auswahl von Interaktionsgeräten mittels Near Field Communication in einem mobilen Bedienszenario im industriellen Umfeld. Kaiserslautern, 2009.

[KLW95] Michael Kifer, Georg Lausen, and James Wu. Logical foundations of object-oriented and frame-based languages. *Journal of the ACM*, 42(4):741–843, 1995.

[KO04] Manasawee Kaenampornpan and Eamonn O'Neill. Modelling Context: An Activity Theory Approach. In Panos Markopoulos, Berry Eggen, Emile Aarts, and James L. Crowley, editors, *Ambient Intelligence*, volume 3295 of *Lecture notes in computer science*, pages 367–374. Springer Berlin / Heidelberg, 2004.

[KVH+01] Lalana Kagal, Vlad Korolev, Harry Chen, Anupam Joshi, and Timothy Finin. Centaurus: A Framework for Intelligent Services in a Mobile Environment. In Makoto Takizawa, editor, *Proceedings of the 21th International Conference on Distributed Computing Systems Workshops, 16 - 19 April 2001, Mesa, Arizona*, pages 195–201. IEEE Computer Society, Los Alamitos, Calif., 2001.

[LC04] Kris Luyten and Karin Coninx. *Dynamic User Interface Generation for Mobile and Embedded Systems with Model-Based User Interface Development.* PhD Thesis, transnationale Universiteit Limburg, 2004.

[LHL01] Berners Lee, J. Hendler, and O. Lassila. The semantic web. *Scientific American*, pages 34–43, 2001.

[Lin22] Linuxnet.com. MUSCLE - Linux Smart Card Development, Last checked: 2010/10/22. `http://www.linuxnet.com/middle.html`.

[LVM$^+$05] Quentin Limbourg, Jean Vanderdonckt, Benjamin Michotte, Laurent Bouillon, and Víctor López-Jaquero. USIXML: A Language Supporting Multi-path Development of User Interfaces. In Rémi Bastide, Philippe Palanque, and Jörg Roth, editors, *Engineering Human Computer Interaction and Interactive Systems*, volume 3425 of *Lecture notes in computer science*, pages 200–220. Springer-Verlag GmbH, Berlin Heidelberg, 2005.

[Mas08] Oliver Maschino. *A Strategy for Automated Generation of Graphical User Interfaces based on the Useware Markup Language in the Domain of Intelligent Production Environments.* PhD thesis, Technische Universität Kaiserslautern, Kaiserslautern, 2008.

[MDH$^+$03] Jennifer Mankoff, Anind K. Dey, Gary Hsieh, Julie Kientz, Scott Lederer, and Morgan Ames. Heuristic evaluation of ambient displays. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 169–176. ACM, Ft. Lauderdale, Florida, USA, 2003.

[Mer07] Merriam-Webster Inc. Merriam-Webster Online Dictionary, Last Checked: 2010/10/07. `http://www.m-w.com`.

[MG08] Gerrit Meixner and Daniel Görlich. Aufgabenmodellierung als Kernelement eines nutzerzentrierten Entwicklungsprozesses für Bedienoberflächen. In *Workshop "Verhaltensmodellierung: Best Practices und neue Erkenntnisse", Fachtagung Modellierung. Berlin.* o.A., 2008.

[MIF25] MIFARE.net. MIFARE 4k Product Homepage, Last checked: 2010/10/25. `http://mifare.net/products/smartcardics/mifare_standard4k.asp`.

[MJM09] Nazir A Malik, Muhammad Younus Javed, and Umar Mahmud. Estimating User Preferences by Managing Contextual History in Context Aware Systems. *Journal of Software*, 6(4):571–576, 2009.

[MUS07] MUSIC Project. MUSIC Project, Last Checked: 2010/10/07. `http://www.ist-music.eu/`.

[OA97] Pinar Öztürk and Agnar Aamodt. Towards a Model of Context for Case-Based Diagnostic Problem Solving. In *Proceedings of the First International and Interdisciplinary Conference on Modeling and Using Context (CONTEXT-97), 4-6 February, 1997, Rio de Janeiro, Brazil*, pages 198–208, 1997.

[Obj08] Object Management Group. Unified Modeling Language, Last checked: 2010/09/08. `http://www.uml.org/`.

[OBS99] M. A. Olson, K. Bostic, and M. Seltzer. Berkeley DB. In *Proceedings of the FREENIX track, 1999 USENIX annual technical conference, June 6 - 11, 1999, Monterey, California, USA*, pages 183–192, 1999.

[Ope08] Open Mobile Allicane (OMA). UAProf Specification, Last checked: 2010/09/08. `http://www.openmobilealliance.org/tech/affiliates/wap/wap-248-uaprof-20011020-a.pdf`.

[ORM08] ORM Foundation. Object Role Modeling, Last checked: 2010/09/08. `http://www.ormfoundation.org/`.

[Pac22] PaceBlade Technology. Paceblade, Last checked: 2010/10/22. `http://www.paceblade.com/site/desktopdefault.aspx`.

[PLV01] Costin Pribeanu, Quentin Limbourg, and Jean Vanderdonckt. Task Modelling for Context-Sensitive User Interfaces. In Chris Johnson, editor, *Interactive Systems: Design, Specification, and Verification*, volume 2220 of *Lecture notes in computer science*, pages 49–68. Springer Berlin / Heidelberg, 2001.

[PMM97] Fabio Paterno', Cristiano Mancini, and Silvia Meniconi. ConcurTaskTrees: A Diagrammatic Notation for Specifying Task Models. In Steve Howard, editor, *Human-computer interaction*, pages 362–369. Chapman & Hall, London, 1997.

[PRS$^+$03] Thorsten Prante, Carsten Röcker, Norbert Streitz, Richard Stenzel, and Carsten Magerkurth. Hello.Wall – Beyond Ambient Displays. In *In Adjunct Proceedings of Ubicomp*, pages 277–278, 2003.

[PSR$^+$04] Thorsten Prante, Richard Stenzel, Carsten Röcker, Norbert A. Streitz, and Carsten Magerkurth. Ambient agoras: InfoRiver, SIAM, Hello.Wall. In *CHI '04 extended abstracts on Human factors in computing systems*, pages 763–764. ACM, Vienna, Austria, 2004.

[Pue97] A. R. Puerta. A model-based interface development environment, Software, IEEE. *Software, IEEE DOI - 10.1109/52.595902*, 14(4):40–47, 1997.

[PvdBW$^+$04] Davy Preuveneers, Jan van den Bergh, Dennis Wagelaar, Andy Georges, Peter Rigole, Tim Clerckx, Yolande Berbers, Karin Coninx, Viviane Jonckers, and Koen de Bosschere. Towards an Extensible Context Ontology for Ambient Intelligence. In Panos Markopoulos, editor, *Ambient intelligence*, volume 3295 of *Lecture notes in computer science*, pages 148–159. Springer, Berlin, 2004.

[RAS08] Carlos Ramos, Juan Carlos Augusto, and Daniel Shapiro. Ambient Intelligence - the Next Step for Artificial Intelligence, Intelligent Systems, IEEE. *Intelligent Systems, IEEE*, 23(2):15–18, 2008.

[Reu03] Achim Reuther. *useML - systematische Entwicklung von Maschinenbediensystemen mit XML, Univ., Diss.–Kaiserslautern, 2003.*, volume 8 of *Fortschritt-Berichte pak*. Techn. Univ., Kaiserslautern, 2003.

[RVDA05] G. Riva, Francesco Vatalaro, F. Davide, and M. Alcaniz, editors. *Ambient Intelligence: The Evolution of Technology, Communication and Cognition Towards the Future of Human-computer Interaction, The Evolution of Technology, Communication and Cognition Towards the Future of Human-computer Interaction.* IOS Press, 2005.

[RWK$^+$08] Roland Reichle, Michael Wagner, Mohammad Ullah Khan, Kurt Geihs, Jorge Lorenzo, Massimo Valla, Cristina Fra, Nearchos Paspallis, and George A. Papadopoulos. A comprehensive context modeling framework for pervasive computing systems. In René Meier, editor, *Distributed applications and interoperable systems*, volume 5053 of *Lecture notes in computer science*, pages 281–295. Springer, Berlin, 2008.

[Rya08] Nick Ryan. ConteXtML, Last checked: 2010-09-08. `http://www.cs.kent.ac.uk/projects/mobicomp/fnc/ConteXtML.html`.

[SAG$^+$93] Bill N. Schilit, Norman Adams, Rich Gold, Michael M. Tso, and Roy Want. The ParcTab Mobile Computing System, 1993.

[SAT$^+$99] Albrecht Schmidt, Kofi Aidoo, Antti Takaluoma, Urpo Tuomela, Kristof van Laerhoven, and Walter van de Velde. Advanced Interaction in Context. In Hans-Werner Gellersen, editor, *Handheld and ubiquitous computing*, volume 1707 of *Lecture notes in computer science*, pages 89–101. Springer, Berlin, 1999.

[SAW94] B. Schilit, N. Adams, and R. Want. Context-Aware Computing Applications. In *Proceedings of the First Workshop on Mobile Computing Systems and Applications (WMCSA), December 8-9, 1994, Santa Cruz, California, USA*, pages 85–90, 1994.

[SBG99] Albrecht Schmidt, Michael Beigl, and Hans-W Gellersen. There is more to context than location. *Computers & Graphics*, 23(6):893–901, 1999.

[SDA98] D. Salber, A. K. Dey, and G. D. Abowd. Ubiquitous Computing: Defining an HCI Research - Agenda for an Emerging Interaction Paradigm, 1998.

[SDA99] Daniel Salber, Anind K. Dey, and Gregory D. Abowd. The context toolkit: aiding the development of context-enabled applications. In Marian H. Williams, editor, *The CHI is the limit*, pages 434–441. ACM, New York, NY, 1999.

[SKD$^+$10] M. Seissler, K. Breiner, P. Diebold, C. Wiehr, and Gerrit Meixner. SmartMote - Ein HCI-Pattern-basiertes universelles Bediengerät für intelligente Produktionsumgebungen. In *Proceedings of USEWARE 2010. USEWARE - Nutzergerechte Gestaltung technischer Systeme (USEWARE-2010), October 13-14, Baden-Baden, Germany.* VDI Wissensforum, 2010.

[SLP04] Thomas Strang and Claudia Linnhoff-Popien. A Context Modeling Survey. In *Workshop on Advanced Context Modelling, Reasoning and Management, The Sixth International Conference on Ubiquitous Computing (UbiComp), 2004, Nottingham ,England*, 2004.

[SLPF03] Thomas Strang, Claudia Linnhoff-Popien, and Korbinian Frank. CoOL: A Context Ontology Language to Enable Contextual Interoperability. In Jean-Bernard Stefani, editor, *Distributed applications and interoperable systems*, volume 2893 of *Lecture notes in computer science*, pages 236–247. Springer, Berlin, 2003.

[SRP+03] Norbert Streitz, Carsten Röcker, Thorsten Prante, Richard Stenzel, and Daniel van Alphen. Situated Interaction with Ambient Information: Facilitating Awareness and Communication in Ubiquitous Work Environments. In *International Conference on Human-Computer Interaction (HCI International 2003)*, 2003.

[Sta08] Stanford Center for Biomedical Informatics Research. Protégé ontology editor, Last checked: 2010/09/08. `http://protege.stanford.edu/`.

[Tec16] Technologie-Initiative SmartFactoryKL e.V. SmartFactory-KL, Last checked: 2010/10/16. `http://www.smartfactory.de/`.

[TLG07] Martin Tomitsch, Andreas Lehner, and Thomas Grechenig. Towards a Taxonomy for Ambient Information Systems. In *Proceedings of the Workshop for Ambient In Systems at the 5 th International Conference on Pervasive Computing (PERVASIVE 2007*, 2007.

[Tra09] Marcus Trapp. *Generating user interfaces for ambient intelligence systems, Introducing client types as adaptation factor, Univ., Diss.–Kaiserslautern, 2008.*, volume 26 of *PhD Theses in experimental software engineering*. Fraunhofer IRB-Verl., Stuttgart, 2009.

[Ubi22] Ubisense - Ubisense, Last checked: 2010/10/22. `http://www.ubisense.net/en/`.

[Use19] Useware Forum, Last checked: 2010/10/19. `http://www.uni-kl.de/pak/useware/index_engl.html`.

[WB95] Mark Weiser and John Seely Brown. Designing calm technology. *PowerGrid Journal*, 1, 1995.

[Wei20] Mark Weiser. Ubiquitous Computing, Last checked: 2010/10/20. `http://www.ubiq.com/hypertext/weiser/UbiHome.html`.

[Wei91] Mark Weiser. The computer for the 21st century. *Scientific American*, 265(3):66–75, 1991.

[WFRS07] Maik Wurdel, Peter Forbrig, T. Radhakrishnan, and Daniel Sinnig. Patterns for Task- and Dialog-Modeling. In Julie A. Jacko, editor, *Interaction design and usability*, volume 4550 of *Lecture notes in computer science*, pages 1226–1235. Springer, Berlin, 2007.

[WHFaG92] Roy Want, Andy Hopper, Veronica Falcão, and Jonathan Gibbons. The active badge location system. *ACM Trans. Inf. Syst.*, 10(1):91–102, 1992.

[Wor07] World Wide Web Consortium (W3C). Standard Generalized Markup Language (SGML), Last Checked: 2010/09/07. `http://www.w3.org/MarkUp/SGML/`.

[Wor08a] World Wide Web Consortium (W3C). Composite Capabilities / Preference Profiles, Last checked: 2010/09/08. `http://www.w3.org/Mobile/CCPP/`.

[Wor08b] World Wide Web Consortium (W3C). Extensible Stylesheet Language Transformations, Last checked: 2010/09/08. `http://www.w3.org/TR/xslt20/`.

[Wor08c] World Wide Web Consortium (W3C). Resource Description Framework, Last checked: 2010/09/08. `http://www.w3.org/TR/rdf-primer/`.

[Wor08d] World Wide Web Consortium (W3C). XML Path Language, Last checked: 2010/09/08. `http://www.w3.org/TR/xpath20/`.

[Wor08e] World Wide Web Consortium (W3C). XML Query, Last checked: 2010/09/08. `http://www.w3.org/XML/Query/`.

[WZGP04] Xiao Hang Wang, Da Qing Zhang, Tao Gu, and Hung Keng Pung. Ontology Based Context Modeling and Reasoning using OWL. In IEEE Computer Society, editor, *Proceedings of the Second IEEE Annual Conference on Pervasive Computing and Communications Workshops*. IEEE Computer Society, Los Alamitos, Calif., 2004.

[Xer13] Xerox. PARCTab Picture Gallery, Last checked: 2010/10/13. `http://nano.xerox.com/parctab/pics.html`.

[Zö9] Detlef Zühlke. SmartFactory - A Vision becomes Reality. In *Keynote Papers of the 13th IFAC Symposium on Information Control Problems in Manufacturing (INCOM 09). IFAC Symposium on Information Control Problems in Manufacturing (INCOM-09), 13th, June 3-5, Moscow, Russian Federation*. ICS / RAS, 2009.