Privacy-Preserving Framework for Context-Aware Mobile Applications

I n a u g u r a l d i s s e r t a t i o n
zur Erlangung des akademischen Grades eines Doktors der
Wirtschaftswissenschaften der Universität Mannheim

vorgelegt
von
Thomas Butter
aus Heidelberg

Dekan: Prof. Dr. Hans H. Bauer
Erstberichterstatter: Prof. Dr. Armin Heinzl
Zweitberichterstatter: Prof. Dr. Martin Schader
Tag der mündlichen Prüfung: 30.09.2009

Contents

List of Figures

List of Tables

Acknowledgements

I would like to express my gratitude to all those who gave me the possibility to complete this thesis. First of all, I want to thank Prof. Dr. Armin Heinzl and Prof. Dr. Martin Schader for their guidance and the freedom I received through my studies. They created a productive environment with great colleagues and much to learn.

I also would like to thank Prof. Dr. Franz Rothlauf and Dr. Markus Aleksy for their good advice and support during the time of my thesis.

Furthermore I want to thank the participants of the Mobile Business Research Group and all my other colleagues for the opportunity to work with them.

My special thanks goes to my family and friends whose patience and support were essential for the completion of this thesis.

1. Introduction

1.1. Problem Outline

In recent years, the pervasiveness of mobile devices, especially mobile phones and personal digital assistants (PDAs), has increased rapidly. At the same time, wireless communication networks have improved considerably and the usage of mobile devices to access the internet is, with decreasing costs, possible almost everywhere and at any time in industrialized countries. However, the usage of mobile technology and mobile applications to support business processes, transactions, and personal tasks is still low compared to their potential.

The improved capabilities resulted in the introduction of many applications for mobile devices by network operators and software vendors. These services were meant to increase the average revenue per user (ARPU) on top of the voice call income. But many of these services have failed and none of them has led to an improved usage of mobile services today, besides e-mail. A new kind of application, the context-aware application, exploits the ubiquity of the mobile devices in order to fit the personal need or task the user is about to execute satisfactorily. Context-aware systems try to improve the communication with the user by adding information about the current context to the explicit user input and by adapting the output to the current setting of the user. While those applications are seen as important steps to a widespread usage, there are strong factors inhibiting their development and adoption.

First of all, the lack of common frameworks handling context data and improving software development increases the cost to build context-aware applications. Each application currently implements its own sensors and logic to handle its data. Furthermore, service providers need to offer tailored services for every context of the user. Since no single provider is able to be an expert for all kinds of applications and will not have the necessary number of developers, a com-

mon service which finds services of multiple providers for the current situation of the user is needed. All services need to utilize the context attributes which are locally determined by the user's situation. Development costs are further boosted by the difficulty of developing applications for multiple devices with varying input/output (IO) capabilities like speech output, small and big screens, full qwerty-keyboards, touchscreens, or numeric keypads.

From the user's perspective, privacy also endangers the adoption of mobile services. Context information may include very private data and expose the user's preferences and habits. While the user may trust a single, well-known, provider to secure the private data and to respect the user's privacy concerns, the problem increases with more and more smaller service providers.

1.2. Objectives

This work examines existing approaches for context-aware mobile applications and develops new ones in order to build a unified framework for mobile services, fostering software development, and to offer a high degree of privacy protection. The framework offers functionality to discover new services, an adaptable user interface (UI) which automatically adapts to the user's context using a UI description based on well-known standards and isolates chosen services for data protection. This offers new and cost-saving ways to develop applications featuring a better adaptation to the user's situation.

Figure 1.1 shows the correlation of usable context data and adaptability of an application. The left graph indicates that the adaptation increases with the amount of available context data, but the available data on the device is not always equal to the data a user is willing to expose to a service provider. Privacy protection leads to a decrease in context data which is available to a service provider. The level of privacy is individually different for each user denoted by the thick red region. This work aims to improve the adaptability while decreasing the exposure of private data to service providers.

The adaptation in this thesis uses a context model that offers a separation of private data, which may never leave the device of the user and public data that may be used by remote services. It will be shown that through the combination

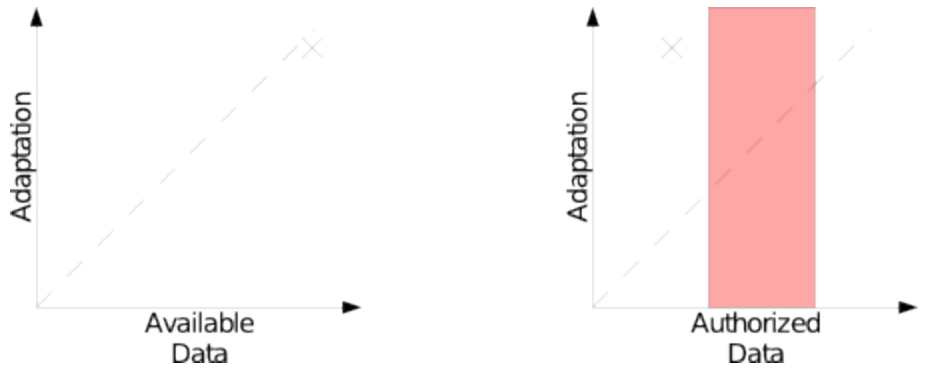

Figure 1.1.: Adaptation vs. Available Data

of local and remote processing of search queries, a context-aware service discovery will be possible without disclosing private data to the search providers.

Research Questions To reach the above-mentioned objectives, in this thesis, the following research questions will be examined and answered:

- Which tasks and responsibilities of a context-aware application are common and can be fullfiled by a suitable framework?
- How is it possible to generalize and improve the service discovery under the aspect of privacy protection?
- How can the heterogenity of mobile devices be handled by developers, especially for UI design?
- How shall applications be designed for the use on different mobile client devices?
- How is it possible to separate different components and protect private date without limiting their utility?

1.3. Methodology

This study originates from the field of information systems research. In this discipline, two different (but complementary) foundational methodologies can be

identified: behavioral science and design science (March and Smith 1995). Research following the behavioral science paradigm focuses on the development and verification of theories of organizational behavior in connection with information systems. Its objective is to understand the interactions between people, technology, and organizations in order to be able to increase the effectiveness and efficiency of information systems which manage these relationships. In contrast to this, the purpose of design science is to improve information systems, to solve problems, and to reach some predefined goals. The focus is on the creation and evaluation of new and innovative artifacts, being either constructs, models, methods, or instantiations (implemented and prototype systems). "Artifacts are represented in a structured form that may vary from software, formal logic, and rigorous mathematics to informal language descriptions". A prototypical implementation of the framework was implemented and its parts are tested for functionality and their suitability to the given problem (Hevner, March, Park, and Ram 2004).

After the successful evaluation of the individual parts, the complete framework is tested using an exemplary scenario involving several sample services. Using these demo services, a descriptive evaluation of the whole framework is done.

As a consequence of this classification, this study incorporates the conceptual guidelines for design science approaches stated by Hevner (Hevner, March, Park, and Ram 2004). These guidelines can be summarized by the table 1.1.

This work follows each guideline and therefore can be classified as design science:

- Guideline 1: The result of this thesis is a framework and architectural model for context-aware mobile applications, a method to improve the privacy protection in three relevant areas and a prototype as implemented as instantiation of a developed concept.

- Guideline 2: A relevant business problem is addressed. The problem and its relevance for business is described in chapter 2.

- Guideline 3: The concept is evaluated using the implementation of demo applications in chapter 6 and benchmarking against the requirements is stated in each chapter and chapter 2.

guideline	description
Guideline 1:	Design science research must produce a viable artifact in the form of a construct, a model, a method, or an instantiation.
Guideline 2:	The objective of design science research is to develop technology-based solutions to important and relevant business problems.
Guideline 3:	The utility, quality, and efficacy of a design artifact must be rigorously demonstrated via well-executed evaluation methods.
Guideline 4:	Effective design science research must provide clear and verifiable contributions in the areas of the design artifact, design foundations, and/or design methodologies.
Guideline 5:	Design science research relies upon the application of rigorous methods in both the construction and evaluation of the design artifact.
Guideline 6:	The search for an effective artifact requires utilizing available means to reach desired ends while satisfying laws in the problem environment.
Guideline 7:	Design science research must be presented effectively both to technology-oriented as well as management-oriented audiences.

Table 1.1.: Design science research guidelines (Source: (Hevner, March, Park, and Ram 2004))

- Guideline 4: New methods for the given problem domains are developed, tested, and individually published for peer review in internal conferences/journals.
- Guideline 5: The construction of the artifacts follows software design principles and is based on well known software-development patterns outlined in chapter 2.
- Guideline 6: An extensive search for design alternatives was performed and design alternatives are shown to each decision made within the parts of this framework in chapters 3, 4, and 5.
- Guideline 7: The concepts are communicated using different abstraction levels. A high level outline of the concepts, a prototypical implementation of each part of the framework, and a sample application.

1.4. Structural Overview

First of all, the current economic mobile business landscape is shown and current barriers for context-aware services are analyzed. Continuing from this point, the requirements of a platform for context-aware services with the potential to overcome the adoption barriers are constructed.

Based on these requirements, different techniques from the field of software engineering are evaluated for their suitability to the problem. Subsequently, known techniques are combined with new ideas in suitable places to a framework for mobile context-aware applications which fulfill the requirements.

The next three chapters are structured according to the usage of mobile services as shown in figure 1.2. In chapter 3, a concept for service discovery is introduced (Butter et al. 2006; Aleksy et al. 2006; Butter et al. 2006). Service discovery is an elementary part of context-aware systems and the initial step in most usage scenarios. If a physical service, like a restaurant or shop, is found, the user needs to navigate the point of interest. This is also covered in chapter 3. The next chapter shows a user interface framework for easy creation and context usage under the requirement of low privacy invasion (Butter et al. 2007) including a prototypical implementation and an evaluation. Service isolation and data protection are examined in chapter 5. In order to offer a runtime environment for downloaded service components, a virtual machine approach is presented. It provides full access to any resource while preserving privacy. The chapter includes a prototypical implementation and measurements about its performance.

An evaluation of the overall practicability of the framework is given in chapter 6. Here, one entertainment / information application for consumers and a support system for the construction industry is implemented using the developed framework. The last chapter includes a summary, a conclusion, and an outlook on future work.

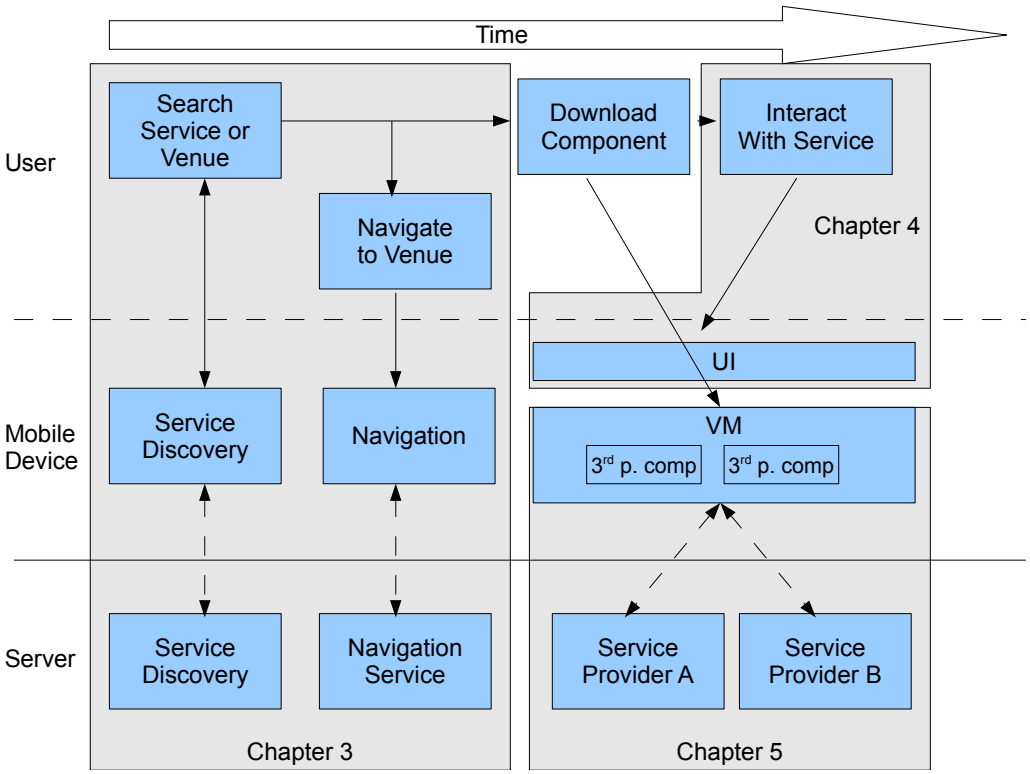

Figure 1.2.: Overview of the Framework

2. Foundations of Mobile Context-Aware Applications

In parallel to continuous miniaturization, as correctly predicted by Moore's law[1] (Moore 1965), mobile devices got increasingly powerful over the last few years. The networks bandwidth increased a lot, too. Gerry Butters stated in Butters' Law of Photonics that the cost to transmit a single bit over an (optical) network halves every nine months, which is equally true for wireless networks. This advance in technology allows hardware manufacturers to build mobile devices such as laptops, personal digital assistants (PDAs), and smart-phones that have a similar performance as desktop computers had a few years ago.

The inherent mobility of these ubiquitous companions leads to new types of applications, customized to the user's location. While location-based services are getting more and more popular, the increasing capabilities of miniaturized sensor technology, which eventually will be embedded in future mobile devices, offer more contextual information, not only location information. Using this additional contextual information within services delivers even more value to the user, leading to higher acceptance and adoption. However, creating context-sensitive applications for mobile devices is not a trivial task and developers' experience with these new kinds of applications is low. Thus, the development of such services and applications or the migration of conventional services to context-sensitive services will require a framework supporting the development of client applications and services.

In this chapter, the foundations for the development of context-aware applications are outlined. The first few paragraphs start with the challenges of mobile applications in general, the following deal with the special requirements of

[1] Moore predicted a doubling in the number of transistors on an integrated circuit every two years.

context-aware applications, and afterwards give an overview of state of the art solutions.

2.1. Mobile Applications

Mobile applications impose specific user requirements and challenges to the developers. The following sections present an overview of those requirements and challenges.

2.1.1. User Requirements

Besides the technical challenges mentioned before, the requirements of the users are an important factor for the functional and non-functional requirements of mobile applications. According to the studies carried out by Bauer et al. (Bauer, Reichardt, and Schuele 2005; Bauer, Reichardt, Exler, and Tranka 2007) and a general literature review, the following features are of special importance:

- Dynamic information is information that changes while the user is on the move. This type of information should therefore be updated permanently and made available to the user in case that his context deems this to be appropriate (Cheverst et al. 2000, p. 2). Examples of this type of information are traffic information, weather forecasts, delayed timetables of public transport, etc. (Kaasinen 2003, p. 74). This requirement implies a constant network connection.

- Customization options for the visualization of information need to be configurable to match the preference of the user, especially due to the limited display space.

- Nowadays, users are flooded with information and sometimes find it difficult to filter the relevant information on many screens filled with data. Services should therefore not intend to provide the user with the most information, but with correct and condensed information (Bieber, Giersich, Kirste 2001, p. 565). An efficient way to improve the efficiency of mobile information services is to adapt the presentation of the services to the current context.

- Transaction security is another major concern of users (Picot, Neuburger 2002, p. 61; Mustafa, Oberweis, Schnurr 2002, p. 367). Secure transactions are a prerequisite for commitment, which is in turn a prerequisite for contracting, which again is an essential for successful business applications (Scheer et al. 2002, p. 101).

- Consumers do also have a great fear which is not connected to any technological concern or the application itself. It is rather rooted in the user's perception of the overall technological advancement that more and more enters his personal life. Many users feel a loss of control over what is happening with and around them and this feeling again evokes great uneasiness. If the application is too always providing the user with the right service at the exact right time, he will develop a feeling of being totally controlled by technology. Users do not want the applications to lead to a predestined and over-controlled environment in which they are completely managed and controlled by their mobile device (Kaasinen 2003, p. 76).

- Consumers generally hesitate to release personal data. They mainly fear the misuse of their personal data in the sense that it is used for purposes the user not explicitly gave permission to (Diezmann 2001, p. 159; M"ohlenbruch, Schmieder 2002, p. 76). Users also fear their personal data being given to governmental institutions and see the "Big Brother" -vision becoming partially true (Pflug, Meyer 2002, p. 413; Turowski, Pousttchi 2004, p. 101).

All these requirements are important for mobile services and are incorporated in the requirements of the framework components in chapters 3 to 5.

2.1.2. Mobile Software Development

In the following sections, technical challenges and general principles of application development for mobile devices are illustrated. The following terms will be used within these sections and shall be defined as follows:

- *Mobile Device*: A mobile device is any electronic device carried occassionally or regulary by the user with input/output capabilities suitable to interact

with the user, a network connection, and the ability to store general purpose software. These are typically mobile phones or PDAs.

- *Mobile Application*: A mobile application is any software executed on a mobile device.

- *Service*: A Service offers a certain resource which may be requested from multiple different users ad hoc. The kind of the offered resource is not relevant and should be considered opaque. A service may be realized by multiple homogenous, parallel running service providers to solve aspects as load-balancing or fault tolerance using replication. Furthermore. it is possible to realize different service providers which offer different Quality-of-Service (QoS) criteria. A service may also be used as a homogenous interface for a subsystem according to the *Facade*-Pattern (Gamma, Helm, Johnson, and Vlissides 1995)

- *Service Provider*: A service provider realizes, at least a part of, a certain service. Using this term, services composed of different components may be better described. According to Shal (Stal 99), a component is a binary, functionally self-contained, software building block which interacts with its environment via a well-defined interface. The component must be able to cope with the embedding in different and non-predictable application contexts.

2.1.3. Technical Challenges

Mobile applications impose several challenges for their development. Especially their smaller size and mobility adds several limitations which have to be worked around:

- *Disconnected operation:* "Mobile computing invalidates some implicit assumptions in current middleware platforms. For example, the CORBA interoperability protocol and webservices assume a permanent transport connection between the client and the object implementation. In contrast to this, PDAs automatically switch themselves off in order to save battery power. [...]Traditionally, we assumed that application entities would be permanently available during some application association" (Geihs 2001).

- *Memory constraints* in mobile devices need new programming frameworks allowing the reuse of already installed components, caching, and eviction (Fahy and Clarke 2004).

- *Flexibility* is important in mobile applications. This is especially the case since the user's current situation decides about the range of useable services. Therefore such applications have to include two special abilities: the ability to efficiently register, discover, use and manage services as well as the ability for ad hoc configuration (Fahy and Clarke 2004).

- *Mobile code* requires increased flexibility and new security mechanisms in order to ensure integrity of the executing machine. (Geihs 2001)

- *Personalization* is important to be able to present information relevant for the user. Customization is a key issue in using mobile devices because it may mitigate some of the limitations of the user interface in terms of size and resolution. Studies show that every additional click reduces the transaction probability by 50% (Durlacher Research, 2000). (Tsalgatidou, Veijalainen, and Pitoura 2000).

- *Heterogenous devices* offer varying sizes, programming environments, input and output capabilities. The applications need to be adapted to each device (see chapter 4).

- *Privacy* is a user requirement and a technical requirement since different components need to be shielded against each other (see 2.3).

Many facets of the first two challenges can be tackled with development patterns known from traditional software engineering. These will be illustrated in this chapter. The others are part of this work's research questions and are dealt with in the chapters 3, 4, and 5.

2.1.4. Runtime Environments

As a example of heterogenity the different runtime environments of mobile devices need special attention. All of them are incompatible and require development from scratch for all supported platforms. The multitude of device manufacturers

and devices offer many runtime environments for applications. Most widely used are the following runtime environments (Köchy 2008):

- *Android* is an operating system based on linux and developed by Google. As of today, two handsets from LG and Samsung are available. It features a Java runtime using an incompatible bytecode format and class libraries for licensing reasons.
- The *iPhone* is a mobile phone from Apple. It is a controlled environment and allows only applications approved by Apple. Currently applications downloading components at runtime are forbidden. (Zdziarski 2008)
- *Windows Mobile* is an embedded variant of Microsoft Windows. Applications in .NET and native applications in C are possible. (Wind, Jensen, and Torp 2007)
- *Symbian* is the most widespread smart phone operating system. It offers an extensive C API and is mainly developed by Nokia. (Harrison and Shackman 2007)
- A *Java* runtime (Java ME) exists on all smart phones, but Android phones and iPhones. The set of supported optional APIs differs widely. (Muchow 2001)

This thesis considers only the Java ME runtimes as those are available on most mobile phones. All concepts would also be possible on Android or other VM based environments.

2.1.4.1. Mobile Java Variants

Java Mobile Edition (Java ME) technology was originally called J2ME and created in order to deal with the constraints associated with small and mobile devices. For this purpose, Sun defined the basics for Java ME technology to fit such a limited environment and make it possible to enable Java applications running on small devices with limited memory, display, and power capacity.

The Java ME platform is a collection of technologies and specifications which can be combined to construct a complete runtime environment specifically to fit the requirements of a particular device.

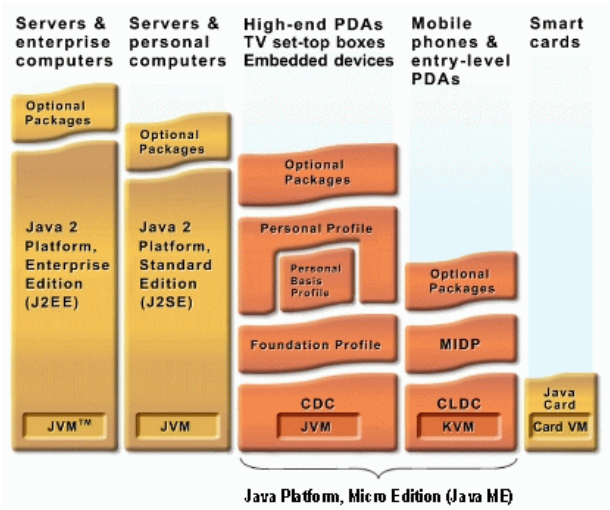

Figure 2.1.: Java Platform

The Java ME technology is based on three components:

- A *configuration* consists of the most basic set of libraries and virtual machine capabilities,
- a *profile* is a set of APIs that support a narrower range of devices, and
- an *optional package* is a set of APIs not available in all devices with a specific profile.

The configuration for small devices is called the Connected Limited Device Configuration (CLDC) (Sun 2004) and the more capable configuration is called the Connected Device Profile Configuration (CDC) (Sun 2005). An overview is given in figure 2.1.

Each configuration, as well as each profile and each optional package is defined through the Java Community Process using specifications called Java Specification Requests (JSR). The configuration targeting resource-constraint devices, such as mobile phones, is called the Connected Limited Device Configuration (CLDC). It is specifically designed to meet the needs for a Java platform to run on devices with limited memory, processing power and graphical capabilities. On top of the different configurations, Java ME platform also specifies a number of profiles defining a set of higher-level APIs that further define the application. A

widely adopted example is to combine the CLDC with the Mobile Information Device Profile (MIDP) in order to provide a complete Java application environment for mobile phones and other devices with similar capabilities. It can be found on the majority of today's mobile devices that access internet services (Admob 2008).

With the configuration and the profiles, the actual application then resides, using the different available APIs in the profile. For a CLDC and MIDP environment, which typically is what most mobile devices nowadays are implemented with, an MIDlet is created. An MIDlet is the application created by a Java ME software developer, such as an entertainment application, a business application, or other mobile features. These MIDlets can be written once and run on every available device conforming with the specifications for Java ME technology. The MIDlet can reside on a repository somewhere in the ecosystem and the end user can search for a specific type of application and have it downloaded over the air to his/her device.

The configuration targeted larger devices with more processing and memory capacity and with a network-connection, like high-end personal digital assistants, is called the Connected Device Profile Configuration (CDC). The goal of the CDC is to leverage technology skills and developer tools based on the Java Platform Standard Edition (SE) and to support the feature sets of a broad range of connected devices while fitting within their resource constraints.

The CDC configuration offers three distinct profiles:

- The Foundation Profile (JSR 219),
- the Personal Basis Profile (JSR 217), and
- the Personal Profile (JSR 216).

For each of these profiles, there is a set of optional packages on which the actual application runs, like additions for media applications, telephony, or location services. On high-end PDA, the Personal Profile is the most widespread variant.

2.2. Context-Aware Mobile Applications

Context-aware mobile applications are mobile applications using context to improve the usability, efficiency, and benefit of a mobile application. The next

sections describe context in general, how context data is gathered, and the privacy issues caused by its usage.

2.2.1. Context in Mobile Applications

In mobile applications, context describes information about the user, the capabilities of the mobile device, or the environment of the user or mobile device. Context can be used for the customization and configuration of services and applications. In literature, there exist many different definitions of context. Most of them define context by listing parameters and categories considered as context or not (Schmidt, Beigl, and Gellersen 1999; Chen and Kotz 2000). The most well-known and used definition can be found in (Dey 2001) who states that "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 the applications themselves."

Displaying complex content on mobile terminals represents a significant challenge. In particular, in the case that one develops a common infrastructure to serve such content to a multitude of different devices. Some of the mobile devices have severe limitations and constraints. Therefore, providing adapted content to each type of terminal requires the consideration of several aspects that are involved in the adaptation and the delivery of the final content. Context information can be used to adapt services or applications automatically and thus can remove unnecessary interactions with the user or decrease media breaks. For example, context can be used to adjust the graphical output of a service to the technical properties of a mobile device by neglecting irrelevant information or by changing the used output device from a text-to-speech device to a screen if necessary (e.g. in noisy environments).

Another important area for context usage is to relieve the user from the explicit definition of context and to automatically generate or derive context information from sensors (Gray and Salber 2001). For example, the context property "user is at work" can be automatically derived from the location of the user. As long as the user is located in or next to his office or working place, the user is at work (with high probability) and it is not necessary for the user to define this context property manually. If the description of the user's situation can be

determined automatically using context information, it is not necessary for the user to define the current situation manually. For instance, when searching for a train connection in a city, information about the current time (important for departure) or the location of the next station can be derived automatically by context information already available and does not need to be specified by the user. Another important aspect is the discovery of new services which may be relevant to the user in his current situation.

Finally, context changes may also be used to trigger actions automatically and to inform the user of newly available possibilities or to start new services. To enable the usage of context information, context providers (e.g. sensors) and the services need to have a common notion for the context attributes used.

2.2.1.1. Categorization of Context

Different categorizations of context attributes exist. The most cited high-level conceptual categorization by Razzaque et al. (Razzaque, Dobson, and Nixon 2005) is shown in Figure 2.1.

Category	**Semantics**	**Examples**
User context	Who?	User's Profile: identifications, relation with others, to-do lists, etc
Physical context	Where?	The Physical Environment: humidity, temperature, noise level, etc
Network context	Where?	Network Environment: connectivity, bandwidth, protocol, etc
Activity context	What occurs, when?	What occurs, at what time: enter, go out, etc
Device context	What can be used?	The Profile and activities of Devices: identifications, location, battery lifetime, etc.
Service context	What can be obtained?	The information on functions which the system can provide: file format, display, etc.

Table 2.1.: Conceptual Categorization of Context
Conceptual Categorization of Context (source Razzaque, 2005)

Most categorizations are similar, but sometimes have slightly different category boundaries. The common categories among those are listed below. Another way to categorize the attributes is the differentiation by data source. This will be shown in the section afterwards. The most relevant types of context are:

Environmental and Network Context Environmental context combines physical and network context and is everything that describes the situation of the user but is not to be directly caused by the user or his mobile device. Common environmental attributes are for example constructed from nearby objects, weather, or traffic (Schiller 2004).

Some context-aware systems generate environmental context information from the position of the user and a projection of the real world into a virtual world. Therefore, nearby objects can easily be found using the location of the user. An example is determining the location of nearby stations, restaurants, or taxis. Other context-aware systems find nearby objects and environmental conditions by using local sensors, such as RFID readers and tagged objects (Römer, Schoch, Mattern, and Düben-dorfer 2003). Frequently, environmental changes occur fast and they are used for triggering automatic actions for the user. However, such changes are difficult to determine reliably (compare 2.2.3.1) and the reliability of such changes should be taken into account before triggering any action based on environmental context.

Device Context Capabilities of mobile devices are valuable criteria for the adaptation of applications to the technical properties of the mobile device. Such parameters do not change as often as environmental context parameters since the device is changed seldom. Relevant context parameters are for example resolution, display size, input and output capabilities and the capabilities of a web browser. These may restrict or forbid the use of some services or applications entirely and therefore may be used for the selection of suitable services for a user.

There are also dynamic properties in this category such as the currently used network connection, the available bandwidth, or the latency and type of payment for the current network connection. Changes in these context attributes can for example be used to trigger automatic synchronizations when a cheap network connection (flatrate) is available or to use graphics with decreased quality in case that only a low bandwidth is available or network traffic is expensive.

User Context The current activity and the role of the user are described by the user status. This context information contains attributes such as work status (leisure time versus work time) or if the user is attending an event (for example a conference). Furthermore, information regarding the role of the user is important

(for example, there is a large difference between the user only attending a talk and the user giving the talk). Such context attributes often determine the availability of the user and can be used to change the types of event notifications or restrict services by the time of availability. An example is to switch off graphical event notifications if the user is involved in a presentation (which can be determined from the user's calendar).

Profiles Profiles are often seen as part of the user context, but consist of explicitly entered information by the user. Profile information is user-specific and is pre-specified by the user. Such context information determines the way a user wants to interact with his mobile device or properties of the user. This information can be synchronized between multiple devices or stored on a server. Profile information is the most reliable and important source of context information, because the user enters this information directly. However, as profiles contain private and sensitive information, it is important to protect it and to allow users to restrict their usage to some services and applications (Jarvenpaa, Lang, Takeda, and Tuunainen 2003). There are three basic categories of profile information. The first are preferences/likings of the user. Personal information like gender, age, and family status constitute the second group. Furthermore, relationships to other users may be used together with their status information which is the core of instant messaging systems such as ICQ, Skype, and others.

- Preferences: The personal preferences/likings are the most explicit information and may be used directly for services or applications.
- Personal Information: Personal information such as gender, age, and family status.
- Relationships: The user's connections to other users (relationships) can be used by context-aware services and applications in conjunction with the status of related persons or users. Relationship information is at the core of messenger systems such as ICQ and others.

History Previous decisions of the user are also important context that can be used for the customization of services. Therefore, it is useful to save the history of

user decisions and to use this information for forecasts, determination of context, or to predict user behavior (Dey 2001).

2.2.2. Context Sensors

Context sensors are the interface from an application to the real world. They may exist directly within the mobile device or may be located on a server which is queried over the network. The latter one offers access to data of other users, but also involves privacy concerns.

2.2.2.1. Local Context Sensors

Several local sources for context information exist which offer some valuable context information. The next sections describe some of them.

Position Location is generally seen as the most important context for mobile services (Cheverst, Davies, Mitchell, Friday, and Efstratiou 2000). Several methods exist to determine the user's position. The most prominent ones are:

- *Global Positioning System (GPS)(Kaplan and Hegarty 2005)* is a network of at least 24 satellites. Each transmits the current time (using high accuracy atomic-clocks) and the positions of the satellites. The receiver uses the signal of at least four different satellites to compute the current time, longitude, latitude, and altitude. It was introduced in the late 80s by the USA Department of Defense and offers a slightly degraded signal to commercial users (about 5m under good conditions). Many mobile devices today include GPS chips, but position acquisiton needs a direct line of sight to the satellites. Using GPS, no position data is available indoors or under a cloudy sky.

- *Galileo* is the European system similar to GPS. It will offer a higher accuracy and should be fully operational in 2013.

- *Cell-based techniques* use the infrastructure of GSM and UMTS networks. It is possible to find the approximate position of the user by looking which antenna and base station he or she uses. Improvements in accuracy are possible by measuring the time a signal needs from the handset to station

(or in the other direction) using different synchronization techniques. An overview of GSM-based positioning systems can be found in Varshavsky et al. (Varshavsky, Chen, de Lara, Froehlich, Haehnel, Hightower, LaMarca, Potter, Sohn, Tang, , and Smith 2006).

- *802.11-based positioning systems* The most promising 802.11-based positioning systems utilize the so-called *fingerprint* approach (e.g., (King, Haenselmann, and Effelsberg 2007)). This technique comprises two stages: An offline training phase and an online position determination phase. During the offline phase, the signal strength distributions are collected from access points at pre-defined reference points in the operation area. They are stored in a table together with their physical coordinates. An entry in this dataset is called a fingerprint. During the position determination phase, mobile devices sample the signal strengths of access points in their communication range and search for similar patterns in the fingerprint data. The best match is selected and its physical coordinates are returned as the position estimate. With this method, a position accuracy of up to two meters is achievable under good conditions.

Calendar Databases may be a source for information about available free time, the next location and the current task of the user. This information may especially be valuable to find a service supporting the user to find a route to its next destination.

Output Capabilities Tailoring the presentation of an application needs information about the output capabilities of the device. This data got increasingly dynamic in recent years. Some mobile phones are now equipped with TV outputs, are able to communicate with video projectors over WiFi, or may use heads-up displays in cars.

2.2.2.2. Remote Context Sensors

Using a remote connection, several additional context sources are available. While these do potentially offer important data, they often pose a problem for privacy protection. This issue is discussed in more detail in chapter 3.

Weather While the weather is easily observable by the user, it is hard to "measure" it on a mobile device which may potentially be in a pocket. Therefore. a remote service may query a weather database using the user's position and return the current weather or a forecast.

Company Schedules In a business environment, it is important which meeting rooms are occupied and which other people are available. Furthermore, social graphs may be used for further inference about likeliness of conversations or the need to meet.

Traffic and Timetables Calculating available time slots involves the place of the next meeting and the optimal routes under the current conditions. The quality of the data may be increased with current traffic or delay data.

Context Sharing Scheduling of ad hoc meetings does not only need the position and the task of one user, but also of the colleagues or friends. This information may be stored centrally on a server (Grossmann, Bauer, Hönle, Käppeler, Nicklas, and Schwarz 2005) if all users trust the provider of this service. In contrast to that, peer-to-peer exchange of context information is possible with several mechanisms. One possibility is the use of instant messaging applications as transport for context data (Leiner 2006).

2.2.3. Quality of Context Data

As mentioned in the previous section, there are multiple sources of context information. Most of these sources have some technical limitations of accuracy and resolution or are based on inference from other context sources which in turn may degrade the resolution and trustworthiness of context information (Buchholz, Küpper, and Schiffers 2003).

From a user's point of view, the use of inaccurate or wrong context information is a larger problem than neglecting the context data entirely. Furthermore, derived context attributes rely on a minimum quality of other context attributes they are based on. For example, when determining context information about the role of a user in a presentation (e.g. to determine the type of user notification), the accurate measurement of the user's position can be important. It can

be assumed that the user is currently speaking if his position is next to the presentation devices. In constrast, it can be assumed that he is only listening to a presentation in the case that he is constantly located more than five metres away from the presentation devices. Therefore, an accurate measurement of the user's position is important for determining the derived context information "user is giving/not giving a presentation". In contrast, when navigating the user to the room for the presentation, an accuracy of five metres may be precise enough to guide the user to the conference room.

New context information can also be derived from other context attributes by inference (Shehzad, Ngo, Pham, and Lee 2004). For example, a context attribute containing the current city can be derived from the geographical location of the mobile device and a city database. For determining new context information, the reliability and accuracy of the used information context is important. When determining the city from the user's position, determining the user's position with low accuracy near a city border would lead to a low probability value for the city.

An important issue of context information is private and public context information. Usually, the user wants to restrict the usage of private context information by the service or application provider. Common ways to restrict the usage is to either not give context information to the service provider or to reduce the quality of the context information that is given to the service provider ("blurring").

2.2.3.1. Quality Losses due to Measurement

The context information accuracy loss can be divided into losses due to technical limitations or deliberately introduced errors to protect privacy. In general technical losses are dominant because the others would be stopped by the users if quality deteriotes too much. For example, GPS devices for determining the user's location have an accuracy of about 15 metres in open field while having a much worse accuracy within cities with limited sky visibility or skyscrapers blocking the line of sight to the satellites. Therefore, it is important for services and applications that make use of context to know about the quality of the context information. Determinants of quality of context information are accuracy, reliability, and time lag.

Accuracy The resolution or accuracy of context information usually depends on the technical capabilities of the sensors. In the simplest form, accuracy can be expressed by a confidence interval, which is a range of values where the correct value is inside with a certain high probability. Depending on the type of context matching, a context-aware system could then consider this information by either assuming the center of the interval as the correct value or ignoring it entirely. An advanced form for considering inaccurate context information is to use probability distribution functions. However, such functions are not available for every measurement tool and difficult to evaluate.

Reliability Most context attributes (especially if expressed as Boolean values) have a level of reliability. For example, the reliability of the context information "user is / is not attending a presentation" depends on other context information and may be more or less reliable. To deal with the reliability of context information, probability values can be assigned to context information in order to be able to consider the trustworthiness during context evaluation. Trustworthiness also plays a role when data entered by other users is used as context information. Here, an anticipated probability of the other user's trustworthiness can be helpful for deciding if the value is important.

Time Lag Not every context sensor may be able to constantly supply all relevant context data at all times. Therefore, context information can be outdated or change over time. To solve problems with time lags, the time of context acquisition should be explicitly noted and transmitted to better utilize the context information. Combining time lag of context sensors with the history of context changes can be helpful for making use of context information with time lag.

2.2.4. Context Representation

Different approaches are used to present context data and to model the context of a user.

Key-value models The easiest form of context representation is the storage in map with values for pre-defined keys. For instance, the set of coordinates could be the value for the key "User Location". These models do not store any kind of

meta data like the accuracy of the values and they do not offer structured data which represents compound data.

Markup scheme models A widely known markup language is the eXtensible Markup Language (XML) which is derived from the Standard Generic Markup Language (SGML). Semi-structured models usually use XML. A characteristic of this model are hierarchical data structures(Strang and Linnhoff-Popien 2004). For instance, one might create the following markup to model the context attribute "network bandwidth":

```
<context>
    <network>
       <attrib name="bandwidth" type="mbit">10</attrib>
    </network>
</context>
```

In contrast to the key-value based models, those models are able to specify meta-data like types (the *type* attribute in the above example) or dependencies(Indulska, Robinson, Rakotonirainy, and Henricksen 2003). Furthermore, the XML documents can be validated against a schema which eases the verification of the data.

Object-oriented models Encapsulation in software engineering ensures that operations on and access to data are performed through a well-defined interface, shielding complex internal workings from the outside. Inheritance means that more specialized functionality can be derived from a set of basic functionalities by simply adding new or modified parts while keeping the parts that are common to the specialized and the general case. These concepts facilitate reusability. Object-oriented context models try to apply the same principles in order to manage the complexities of representing and handling context information. In that respect, they succeed quite well as they can easily be implemented using today's computing systems and allow for the inclusion of dependencies and meta-information. Using object-oriented programming languages, providing meta-information as well as defining and listing dependencies, is mostly a question of designing suitable interfaces for context-handling classes(Strang and Linnhoff-Popien 2004).

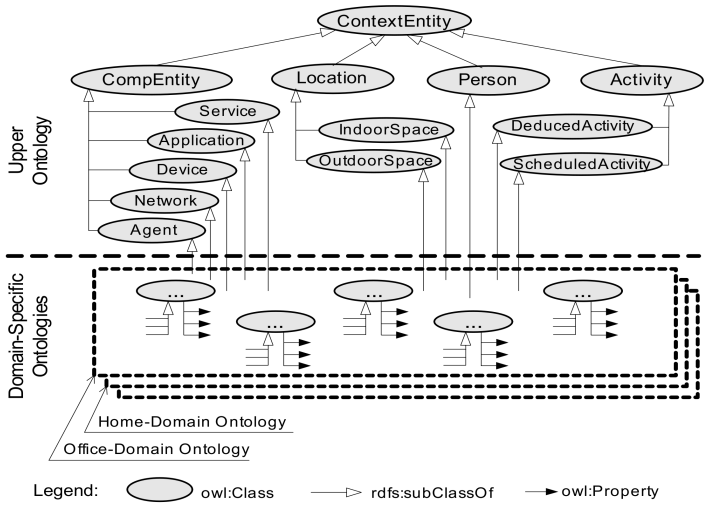

Figure 2.2.: Partial Definition of a Context Ontology, source: Wang (2004)

Logic based models Context can also be expressed in highly formalized logic based models by using facts, rules and expressions. However, logic based models are not very useable on mobile devices since they require a high complexity in their evaluation (Strang and Linnhoff-Popien 2004). General logic reasoning is inherently non-deterministic and thus evaluation algorithms do either not guarantee to produce a result in a finite period of time or have exponential runtime and high memory consumption (Schöning 2000).

Ontology based models Originally, ontologies have their seeds in philosophy which have been coined differently in the field of computer and information science(Floridi 2003). In computer science, ontology refers to "an engineering artifact, constituted by a specific vocabulary used to describe a certain reality, plus a set of explicit assumptions regarding the intended meaning of the vocabulary words" (Guarino 1998).

An examplary meta model is shown in figure 2.2. The strengths of ontologies are:

- power of expression: context can be formally modeled as a set of concepts and their relations. Subconcepts are capable to define more specific cases of general concepts and context can be enriched using meta-information. In figure 2.3, a subconcept for the example above that shows the more specific home domain is given;

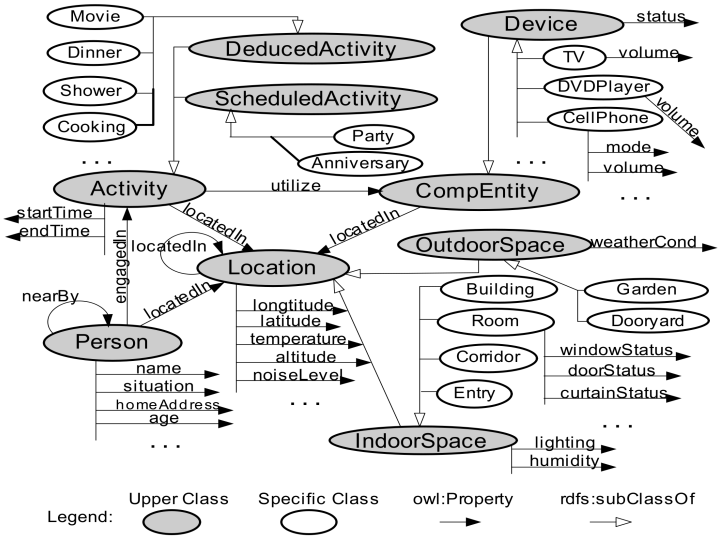

Figure 2.3.: Definition of a Specific Ontology for Home Domain, Source: Wang (2004)

- support for logic inference: higher level context information can be derived from raw context data and context data can be checked for inconsistencies.

Ontologies are limited to specific application scenarios as a universal context ontology that covers every conceivable piece of context information for all aspects just might not be attainable(Wang, Zhang, Gu, and Pung 2004). Also, support for logic inference generally poses the same high processing requirement found with logic based models.

Graphical Models Graphical context models are aimed at improving formality and expressiveness in context modeling in the design phase.

An example of adding quality information to a context model in the form of freshness and accuracy indicators is shown in figure 2.4. Graphical models are not primarily for the actual realization, though some approaches can be adapted quite well to enable context management in database-driven environments, because they are based on entity-relationship diagrams which are commonly used to model databases(Henricksen, Indulska, and Rakotonirainy 2003).

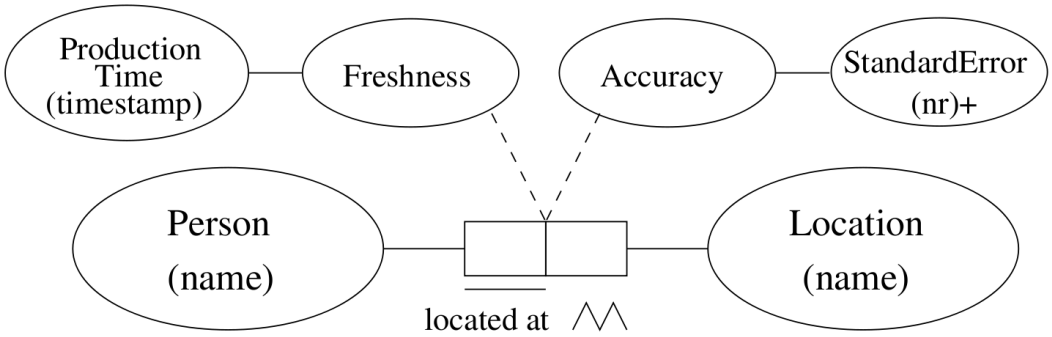

Figure 2.4.: Graphical Context Model, source: Henricksen (2003)

2.2.5. Design Patterns for Development of Mobile Applications

The previous sections outlined the requirements of context-aware mobile applications. Several definitions and classifications of context and different types of context models to describe context were shown. Besides the need for context models several design techniques are needed to overcome the technical limitations of many mobile devices in contrast stationary computers (cf. 2.1.3). Also the high dynamic in usage situations requires special attention. The following sections will show several software design patterns which were choosen as guiding principles to solve some of the technical problems.

Design patterns are simple and concise solutions for common programming problems. Here, they will be used to show the best practices for mobile application development. The architect Christopher Alexander (Alexander, Ishikawa, Silverstein, et al. 1977) is generally seen as the mental father of the design pattern movement. In the area of software engineering, design patterns became popular by the work of Gamma et al. (Gamma, Helm, Johnson, and Vlissides 1995). Their structure follows certain rules and is based on the following elements:

- *Pattern name*: The name of a pattern is usually short, yet descriptive, and acts as an addition to the design vocabulary.
- *Problem*: Patterns should include a short description of the problem they intend to solve.

- *Solution*: The solution to the problem is described in a generally applicable way. The elements of the solution are described along with their relationships and responsibilities.

- *Consequences*: While the main consequence of using the pattern is the solution to the problem, there do often exist side effects. In order to make it easier to understand the trade-off involved in using the pattern, it is important that the potential drawbacks are explained.

As a matter of course, this is not the only possible structure for describing a pattern, in fact Gamma et al. also provide an extended schema (Gamma, Helm, Johnson, and Vlissides 1995) and other authors offer alternative descriptions as well (Fowler 1997).

A design pattern documents a comprehensive solution for common tasks during the program design that evolved through experience into a useable solution for many situations. The engineering task during the object-oriented software design is to find a suitable design pattern for a given programming task or many design patterns which together solve the problem.

The documentation of the design patterns helps in the design process from completing requirements of the given problem to the realization in a programming language. It also acts as a foundation for discussion between the developers about the solution.

The primary use of a design pattern lies in the documentation of a solution for a given class of problems. Design patterns are generally language independent. During the design phase of object-oriented software, they are an accepted tool to foster the design process.

The ability to efficiently discover, manage, and use services is of essential importance for the design of mobile solutions. Hereby many problems have to be solved.

Memory constraints Memory is a scarce resource in mobile devices. While many devices feature more than one gigabyte of flash memory for long-term storage, the amount of random-access memory for executable software is generally lower than 128MB. Among other things, the deallocation of no longer used services is important to deal with this scarcity. This section will show different

design patterns for the implementation of an infrastructure which could be used for mobile clients.

In the case of mobile clients, the use of the *Evictor*-Pattern (Henning and Vinoski 1999; Jain 2001) is the most relevant solution. This approach tries to reach the goal of an efficient use of services through monitoring. Every time a service is accessed, a marker is set. Services which are not used for a long time (least recently used – LRU) or which are used only rarely (least frequently used – LFU) are candidates for deallocation. The deallocation of services may happen periodically or explicit at any time. The pattern involves the following four roles:

- *User* is using the resource
- *Resource* offers a specific functionality
- *Evictor* is the instance which is responsible for the eviction of the resources
- *Eviction Strategy* defines the criteria for the eviction of resources

The main advantage of this design pattern is transparency for the client, i.e. the mobile client does not have to take care about the eviction of his used services. The eviction of no longer needed resources is also possible during technical problems, such as a loss of network connection. The only disadvantage is that the client application has no chance to reject the eviction of a service. This disadvantage is that you need a mechanism to easily to deal with the configuration of the eviction strategy.

Disconnected Operation Another problem is the fact that not every service is always available. This may in turn be caused by technical problems (such as loss of connection) or a changed location. In this case, the *Leasing*-Pattern (Jain and Kircher 2000a) could be used. This pattern is based on a method that does not offer a service for an unlimited time, but only for a limited period. When the time is over, there are different possibilities. The leasing-concept, which is used e.g. in Jini (Inc. 2003) and is also specified by Sun, was disregarded for mobile location-based patterns in the past. This concept is not only useful for Jini, as the work of Jain and Kircher (Jain and Kircher 2000a) suggests which even talks about a “Leasing Pattern”. The authors focus on the exact details of the problem, the structure, and their possible variants. In technologies which

do not support the leasing concept directly, such as Common Object Request Broker Architecture (CORBA), the concept can be realized through a special service (Aleksy and Gitzel 2002).

The leasing concept involves four roles:

- *Resource*, which offers a specific functionality
- *Resource Claimant / Holder* is the component willing to use the resource. It may only use the resource if it owns a lease.
- *Lease* is the link between the resource and the resource claimant. It offers the functionality to free a resource prematurely or to extend the leasing time if possible.
- *Lessor / Grantor* is the main management part. It manages the assignment of the resources to the claimants.

Reconfiguration Furthermore, there are some more design patterns which assist in the dynamic reconfiguration of software components, e.g. the *Service Configurator* (Jain and Schmidt 1997) or the *Component Configurator* (Schmidt, Rohnert, Stal, and Schultz 2000). Even more related patterns may be found in Welch et al. (Welch, Marinucci, Masters, and Werme 2002).

The Design Patterns Lazy Acquisition, Virtual Proxy, and Virtual Component may be used in the field of client-development for mobile applications. They especially take things like the conservative use of resources into account and foster the development of adaptable applications.

The second category of the shown Design Patterns is–directly and indirectly–concerned with the client and the service part of the application. For example, the Lookup Pattern works as a link between the client part of a mobile application and the (potentially) used services. The use of the Evictor Patterns frees the mobile client indirectly from the use of elaborate and complex algorithms for the efficient utilization of resources.

Other Design Patterns, i.e. Component Configurator or Service Configurator, are concerned with the configuration of services and may lead to a higher flexibility.

Discovery The following paragraphs introduce different design patterns which can be used for the discovery of available services. First aspects of the client applications are examined. Later on, the necessary approaches for the server-based infrastructure are shown.

To utilize a service, a mobile client needs to have the ability to discover available services in a dynamic environment. The offered services could differ depending on the context of the user. This brings in the necessity to decouple the client from the services.

A possibility to solve the first problem would be to let every service periodically broadcast a beacon and make itself known to the clients and thus its potential users. An alternative would be to let every client announce its availability via a broadcast message and every available service sends a reply.

Both techniques are not scalable in the internet and any other unicast network since the broadcast traffic increases with every user and service.

One common solution to this is the *Lookup*-Pattern (Jain and Kircher 2000b). It involves three roles:

- *Client*, which uses a service,
- *Service* offering a certain functionality,
- *Lookup Service*, which allows other services to register. Clients could use this service to find other services.

This approach is used in different popular technologies, such as the Common Object Request Broker Architecture (CORBA) (OMG 1995). Further patterns concerning service discovery—more exactly a complete pattern language—are described in (Pärssinen, Teemu, and Eronen 2005).

Flexibility The dynamic behavior of mobile applications is also a challenge for developers. This means that they should not be coupled directly with a service. To decouple the possible heterogeneous mobile clients from the services, the *Service Abstraction Layer*-Pattern (Vogel 2001) may be used. This design pattern adds another layer between the client-applications and the services. This may be realized using the *Facade*-Pattern (Gamma, Helm, Johnson, and Vlissides 1995). The Service Abstraction Layer boosts aspects as separation of concerns, generic request handling, controlled evolution, and communication transparency.

In this way, it is possible to tailor services to client applications which are specific to a location or which have different abilities, such as different Quality-of-Service-Characteristics (QoS).

Through the constantly changing business processes and the ongoing advent of emerging technologies, the "Separation of Concerns" (Vlter, Kircher, and Zdun 2000) principle should get special attention. This is, together with the presented patterns, an integral part for the development of platform- and language-neutral mobile applications.

Another problem, which has to be solved, is the efficient use of the resources of the mobile client device. Despite the amazing technical advancements with respect to memory or available bandwidth, these resources are scarce in mobile devices. At this point, one could use patterns such as *Virtual Proxy* (Gamma, Helm, Johnson, and Vlissides 1995). It creates "proxy" objects which dynamically load "expensive" objects on demand and thus save active memory. For similar reasons, patterns such as the *Lazy Acquisition*-Pattern (Kircher 2001), which aims to acquire memory as late as possible, were created. It is based on the following four roles:

- *User*, who wants to use a specific resource,
- *Resource*, which offers a specific functionality,
- *Virtual Proxy*, which offers the same interface as the resource it represents and functions as local stand-in,
- *Resource Environment*, which is responsible for the management of the resources.

Furthermore, the Virtual Component (Corsaro, Schmidt, Klefstad, and ORyan 2002) helps with the aspects of high configurability and flexibility at run-time. This pattern includes six elements:

- *Component* defines the interface to use the functionality of the Component,
- *Concrete Component* is the concrete implementation of the Component which offers the ability to load only the needed parts of the component,
- *Component Factory* offers an interface and a method to create the component,

- *Concrete Component Factory* is a concrete implementation of a Component Factory which instantiates the Components,
- *Loading Strategy* decides when to load a Component and when to instantiate it,
- *Unloading Strategy* decides when and if a Component and its associated resources should be freed.

The interface of a Component represents a building-block of an application and is realized through a Concrete Component. For the creation of Concrete Component instances, the corresponding factories are responsible which could use different Loading/Unloading-Strategies.

2.3. Privacy in Context-Aware Applications

Context-awareness provides many benefits for the adaptation of an application to the current task for the user. However, with the advent of mobile devices, which are carried by a user all the time and are always-connected, the protection of the user's privacy became an important challenge. While the applications are collecting more and more information from sensors and recording the habits and preferences, the potential for intrusion of the user's privacy becomes even bigger.

As Westin defines privacy as "the claim of individuals, groups or institutions to determine for themselves when, how, and to what extent information about them is communicated to others" (Westin 1967), the user must have the possibility to deny the transfer of personal information.

Many studies show that privacy is one of the main issues users are facing in context-aware systems (Barkhuus and Dey 2003; Bauer, Reichardt, and Schuele 2005). Privacy also has an economic value for both the user of a service and the provider, which shows an incentive to invade the privacy of the user (Varian 2002). The increased availability of private data also increases the risk of identity theft (Berghel 2000).

One way to improve the privacy is to make all outgoing data anonymous on the way to the service provider(Tatli, Stegemann, and Lucks 2006). Here, problems arise when subscription services are used which need some kind of authentication.

Another way to compromise the privacy is the physical theft of a device, so encryption of personal data is needed.

Besides protecting information, there needs to be a way for users to decide which pieces of information may be shared with whom. With a large number of service providers and peer-to-peer users, there needs to be a system which is capable of determining the user's preferences from a small set of questions. (Leiner 2006; Hull, Kumar, Lieuwen, Patel-Schneider, Sahuguet, Varadarajan, and Vyas 2004)

Privacy and data security do have different implications for consumers and companies in B2B enviroments outlined in the following sections.

2.3.1. Consumer Privacy

Consumers often do not see the problems of privacy invasion immediately since they do not have the knowledge how their data can be combined and aggregated. It is often underestimated how easily one can uniquely be identified with only some small amount of data. Sweeney showed with the US Census Data of 1990 that 90% of the US population can be identified with only their gender, ZIP code, and date of birth. With a textual representation of the place (which could be city or town), about 53% are still uniquely identifiable(Sweeney 2000). Golle re-examined the 2000 US Census Data and showed that even if you substite the "place" with the "county" 14.8% are still identifiable and 43.6% are 5-anonymous[2] or less (Golle 2006). The problems often surface after using the services for a while and then a consumer backlash may happen after some negative press coverage. Big social networks collected a lot of data of their users and they were willing to make them available in large quantities. After they started to do targeted advertising using this data, many users began to feel uncomfortable with the large scale data collection (Gross and Acquisti 2005).

Besides the data collection of cooperations, the large-scale surveillance of the citizens by their government creates another privacy risk for people. The government has access to the data of the service providers in many jurisdictions, which is worsened by the multi-national nature of most big cooperations. The extreme cases of surveillance and their consequences are outlined in classic novels. Orwell portrays in "1984" (Orwell 1949) a totalitarian state which monitors all citizen

[2]k-anonymous means that someone is hidden indistinguishably in a group of size k

and enforces the strict rules of the state. In contrast to this, Kafka shows in "Der Process" a state which is not perceived as "bad" by the protagonist, but the vast information collected leads to some false accusations and a trial (Kafka 1925).

These fictitious stories show some of the problems of privacy invasion by states and are in part transferable to large corporation aggregating vast amounts of personal data. A full analysis of these privacy problems and the common argument "I've got nothing to hide" is given by Solove in his article with the same name (Solove 2007). One recent example of government regulated data retention abuse was unveiled in May 2008. The Deutsche Telekom used telephone connection details to spy their supervisory board. This led to an increased awareness by their customers and many stated that they will change their telephony behavior[3].

2.3.2. Data Protection in B2B Environments

The need for data protection in B2B Environments is even more obvious than in B2C environments. Companies do have their sensitive data and do not want to give any valuable information to their competitors. This is especially true in environments with always changing partners. One industry especially interesting for mobile services is the construction industry. The ever changing building site prohibits an extensive use of desktop PCs and a pre-built infrastructure. Also the distribution of workers between many sites complicates the monitoring of tools and workers. Increasing complexity of engineering and construction processes leads to virtual organizations for a single project involving shared processes.

"These processes span enterprises, potentially in different countries or economic zones and are possibly subject to export restrictions. Each partner providing a major sub-system (e.g. engine, fuselage of a civil aircraft) also has a network of component suppliers and logistics support. These production processes are complex and may require re-scheduling in the case of delays in deliveries, etc. This may necessitate the sharing of process information (so-called "process visibility") across the VO, entailing flexible security systems that control access to internal information." (Dimitrakos, Golby, and Kearney 2004).

Notices about delays have to be passed on over organization boundaries but may not reveal too much about the supplier network of each participant. For the varying virtual organizations, different services are needed and therefore it would

[3]Die Zeit, 5.06.2008, page 1

be desirable to easily use components of partners without the possibility of data leakage.

2.3.3. Private and Public Context

In most current mobile applications, all context information is used and transferred to a server that offers the requested information or application. However, when dealing with context information, privacy issues are very important. If privacy issues are not considered appropriately, end-users will not adapt and use new services (Ho and Kwok 2003). Therefore, mechanisms are necessary that allow end-users to control the usage of context by mobile services or applications (Ackerman, Darrell, and Weitzner 2001).

Many context attributes are naturally very personal and breach the privacy of the user considerably. To overcome the user's fears of privacy loss, it must be possible for the user to declare context information as private or public. Public information may be used by service and application providers, but not private information. There are also intermediate states of context information possible by reducing the quality of context information ("blurring"). Then, the user intentionally reduces the quality of context information. A common example is disguising the user's identity when using specific services by using a slightly changed age or location.

Blurred and public context information often includes enough data for the service provider to pre-configure a service while still making it hard to track the exact context of a user. The existence of blurred and private context results in a situation in which the service provider can consider less context information than the mobile device for the customization of services. However, the mobile device often does not have the technical capabilities to perform extensive configurations. Therefore, mechanisms are necessary that consider the tight hardware restrictions of mobile devices and which can use the private or blurred context information for a user-specific configuration of services on the mobile device (Leiner 2006).

When searching for reasons that explain the low usage of advanced mobile applications and services (Ho and Kwok 2003), privacy issues are seen as an important, but often neglected, barrier for a widespread use of such services. Users do not want to give private information like user profiles, information about the environment, or other types of context to a service provider, but restrict or pro-

hibit the usage of such information. Solutions which differentiate between private and public contexts are needed: Using a service or an application, public context information can be sent to a service provider and used for the personalization and customization of the service. In contrast, private information is not sent or substantially degraded in its quality before sending.

Each user chooses which context attributes are private and which may be used outside of his mobile device. Furthermore, he may choose to deteriorate the quality of some attributes in order to decrease the possibility to be tracked by a service provider.

3. Service Discovery

Mobile devices became ubiquitous companions for most people in industrialized countries. Those devices are getting more powerful and they combine ever more functions like a camera, a music player, an organizer, and a telephone. An increasing number of mobile phones includes a bunch of sensors enabling them to know their current location or measure acceleration and gestures.

Using these features, new types of services and applications are possible which assist people in many common situations. Nevertheless, no single mobile "killer application", dramatically increasing usage of mobile services, has been found. Instead people need many services tailored to different usage scenarios, would be able to fulfill together many personal user needs in varying circumstances. So, a bundle of services is more suitable to increase the benefit of the mobile devices (Stremersch and Tellis 2002). Not all necessary services can be anticipated before production or even fit into the memory of a single device. Therefore, an important functionality is the discovery of further suitable services for a user, depending on his location, task, or, more general, his current context. In this case, a service could be a software downloadable to the device, a remote service accessible over the network, or a physical shop or person offering some business service needed by the person to follow his current intention.

In this chapter, a generic search scenario is described. Based on this scenario, the foundations outlined in the previous chapter and the current literature requirements for a discovery service are derived. Special attention will be given on the privacy aspect whose importance was outlined in the previous chapter. It is followed by an overview of existing approaches and an analysis of their quality with regards to the requirements. Afterwards, a concept for a privacy aware service discovery is illustrated and a prototypical implementation is presented. Based on the concept, the suitability of the concept for routing and navigation services (which are similar to a service discovery) is shown. The chapter con-

cludes with an evaluation of the concept based on the evaluation criteria from the requirements.

Illustrating the basic concept for a service searching for restaurants, this example should be used as a representative for discovery services that are initiated by a mobile user and return a list of search results – in this case a list of restaurants or software components matching the current desire of the user. Discovery services are offered by service providers and only the outcome of the search – a list of search results – is sent to the mobile device of a user. Using discovery or search services, usually user-specific context information like location, preferences, time, and other context information, is considered in the search. A sample scenario could look like this:

The mobile device senses the location of the user within a range of several meters, has the profile data indicating that the user is a smoker, and only wants to pay with credit card. The user declares the context information "pay with credit card" as private, meaning that no potentially untrustworthy service provider should know about it. Furthermore, he does not want to give his exact location to the service provider, but reduces the accuracy of the context information "location" to a few hundred meters. The context information "smoker" is used as public context since it is not seen as sensitive by the user. Now, the user wants to the find best suitable services beneficial in his current situation. The main challenge is to find the service without disclosing too much data. A software implementation for this scenario is presented in chapter 6.

Navigation is often linked to service discovery, especially if physical services instead of web services are searched for. Making use of a business service usually requires getting to the physical location of the service. Navigation to the place of service is a consequence if the user does not know the region. Similar to the discovery of relevant services, the routing between the current position and a destination offers many possibilities to infer something about the identity and the context of the user. Even if navigation is performed by a different service provider, it could use the destination in order to limit the number of potential services to a few and the starting location to infer something about the previous activity of the user. Therefore, the current position and the destination has to be hidden and a privacy preserving routing algorithm has to be used.

3.1. Foundations of Context-Aware Service Discovery

Based on the previous chapter and a literature review on user requirements, the following important requirements for a reusable context-aware service discovery were identified as especially important.

Extendable Kaasinen states that "Users are different and they may use the services for many different tasks, even for tasks that were not anticipated in the design" (Kaasinen 2003). Having a pre-defined set of rules to find service types according to a context would set barriers for new usage scenarios or new kinds of services. This excludes a pre-defined set of service attributes which are hard-coded into the mobile application and fixed rules about relation between attributes and a specific service.

Reusable Being part of a framework which should allow creation of arbitrary context-aware applications, the service discovery must be reusable together with a partial concrete implementation of the problem solution (Wirfs-Brock and Johnson 1990). In the domain of a context-aware service discovery, this means that it should be useable to discover different kinds of services in varying application domains. It is therefore linked to the first requirement of extendability.

Privacy In section 2.3, important privacy literature was reviewed. Especially the issues in context-aware service discovery were researched, amongst others, in (Kaasinen 2003): "In our group interviews, people were worried about their privacy and the 'big brother' phenomenon when considering services enabling people to be located". A solution therefore has to respect the privacy of the user and always make the used data transparent to the user.

Learning from previous behavior The personal preferences of each user are different (Bauer, Reichardt, and Schuele 2005) and so may be the individual weight of varying context attributes. A search for services and mobile applications should be able to improve its results over time, based on the user's decisions in the past.

3.2. Existing Approaches

The PARCTab project(Schilit, Theimer, and Welch 1993) research has focused on the subject of context-sensitive (context-aware) service discovery. Other research projects focused on context-sensitive service discovery in earlier times which has been related to hardware, like near-by printers or other low-level services. In this section, a brief overview on related work is given with regard to service discovery using contextual information.

In the CB-Sec project (Mostefaoui and Hirsbrunner 2004), an architecture has been developed focusing on service discovery of web services using contextual information. Therefore, a service description schema has been developed that includes a set of constraints, requirements, and context functions that are used by a brokering agent to evaluate, filter, and rank services which fit a given set of context best. Context is collected by the context gatherer that receives contextual information from software and hardware sensors and is stored over time in the context data base that is available to the whole system. Context attributes that are used for service discovery are, for example, the location of people, device and service capabilities, context history, time of day, and others.

Kuck et al. present an approach for context-sensitive discovery of Web Services that is based on matching the user's context and enhanced service descriptions using information from WSDL descriptions and a UDDI repository. Their service description includes inferred information of syntactical and textual contents of a WSDL description as well as feedback information, e.g. the context at the time of service recommendation (Kuck and Reichartz 2005).

In the COSS approach (Broens, Pokraev, van Sinderen, Koolwaaij, and Costa 2004), ontologies are used for the context and service descriptions. Service advertisement and service requests are represented as documents, while service requests include attributes defined by the user. An attribute like "nearby" is enhanced by rule that is evaluated during the matching process, for example the user's location is within a certain distance to the service's location. In the COSS approach, context providers deliver the necessary contextual information needed for the matching process. Related to the WASP project, a service platform for mobile context-aware applications has been built up (Pokraev, Koolwaaij, van Setten, Broens, Costa, Wibbels, Ebben, and Strating 2005).

3.3. A Novel Service Discovery Concept

All approaches in the previous sections are based on a centralized discovery algorithm which uses all available context information to match them against the available services. The matching services are then presented to the user for further inspection and selection. Depending on the approach, this matching phase is performed on the device of the user or on a server. There are approaches using a hierachical system forwarding a request to the next server depending on a first broad match (e.g. (Grossmann, Bauer, Hönle, Käppeler, Nicklas, and Schwarz 2005)), but there is still one central server which gets all context data at one time. The privacy protection is enhanced by using anonymization which removes the connection of subsequent queries by destroying the session or randomizing the IP address of the connecting device (for an overview see (Tatli, Stegemann, and Lucks 2006)). This certainly decreases traceability for many use-cases, but only works with a high number of users and non-identifying context attributes. Having something like the home address as context attribute allows easy correlation of a request with a specific user. Executing the whole service discovery on the client eliminates the privacy issues, but the number of discoverable services would be limited because of memory and bandwidth constraints.

Goal of this chapter is a concept which combines the large number of available services with the high degree of privacy protection. A solution relying only on the servers or only on the client to run any discovery was shown to be not feasible. Therefore only a distributed approach can be used. The concept shown here distributes the discovery between one or multiple servers and the mobile client to combine the advantages of good privacy protection and an unlimited number of services to choose from. The concept is described in the next section detailing the representation of context data, a context manager to gather and store the data, the discovery server component, and the mobile client component.

In summary, the client sends a representation of the public context attributes to a discovery server. This server then performs a broad matching in order to find a list of potentially suitable services and returns this list together with some rules on how to refine the results using further, potentially private, context attributes. These rules will then be processed by the client component to find the final result. An overview of the process can be seen in figure 3.1. Detailed rationale behind these design decisions is given in the respective sections.

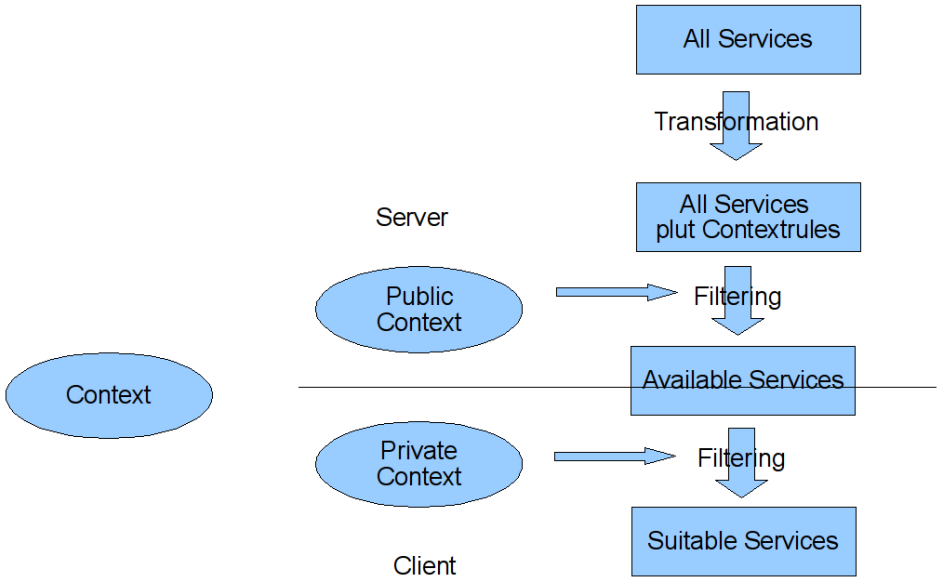

Figure 3.1.: Service Discovery

3.3.1. Context Representation

An overview of different context representations is given in section 2.2.4. In our concept the representation of context data has to be useable for the transmission of data to the server and for the refining rules sent back to the client. The representation may not include any datatypes requiring knowledge besides the ones available at build time, but still has to be extendable to any new context attribute.

Key-value models do not offer any semantic or even metric for the values and therefore can not be used to refine the search on the client side (cf. 2.2.4). Logic or ontology based models would have a very high computational cost on the client.

Therefore a markup-based representation was chosen which is easily extensible and offers four primitive types represented in different forms. Any context attribute has to be expressed using one of these primitives or a combination of them. The primitives were choosen in a way to accommodate a large list of context attributes. For each primitive, a representation for a single value and for a

set or range of values does exist. Using these different representations, the current context as well as the rules to refine the search results can be expressed. These sets or ranges can be used to describe the range for which a service is applicable.

Each context attribute needs a well-known name and has to be expressed using one of the four primitive types. If an attribute is unknown, it is still possible to match its value with a rule only by knowing its type and name.

The primitives are:

- *Coordinates*: a Coordinate is represented in different coordinate systems. The defaults are carthesian coordinates. Sets may be expressed as circles around a point and every point within a polygon.
- *Time*: a Time consists of a date and a time of day. Ranges are expressed as daily, weekly, monthly, or one-time intervals.
- *Number*: a Number is an arbitrary precision decimal number or a range with a lower and upper bound.
- *String*: a String may represent any nominal type. It may match a single String, a set of Strings, or a regular expression.

Table 3.1 gives an overview of the types and their elements in their serialized form.

Primitive	**Type**	**Elements**
Coordinates:	CircleContextAttribute	Center, Radius
	PointContextAttribute	Latitude, Longitude, Altitude
	PolygonContextAttribute	Point (1 or many)
Time:	DailyTimeIntervalContextAttribute	Weekday, StartTime, EndTime
	DateRangeContextAttribute	StartDate, EndDate
	MonthlyTimeIntervalContextAttribute	DayOfMonth, StartTime, EndTime
	SingleDateContextAttribute	Date, Time
	Time	Hour, Minute, Second
Number:	NumberRangeContextAttribute	Start, End
	SingleNumberContextAttribute	Value
String:	SingleStringContextAttribute	Value
	StringSetContextAttribute	Value, Value,...
	StringRegExpContextAttribute	RegExp

Table 3.1.: Primitive Types of the Context Representation

A context attribute using one of these primitive types can then easily be compared to another representation of the attribute in the case that only the primitive

type is the same. Using this scheme two sets of context attributes are easily comparable on the mobile device without a large amount of processing power. Two sets of attributes "match" if and only if

- each attribute name appears in none or both sets,
- those with the same name have the same primary type, and
- the value in one set is contained in the other set's attribute.

An attribute value is "contained" in another one if

- both are an exact value and both values are equal,
- one is a range/region and the exact value is contained in the range, or
- both are a range/region and both overlap.

3.3.1.1. Quality of Context

Any attribute having continuous values may be imprecise. This may be caused either by error of measurement or by purpose to reduce the invasion of privacy. An example for the latter case would be the decreased precision of the current position of the user which is sent to a service. This impreciseness would cause problems for the matching of a region/range and a position/number. A service could be valid in a circle of 1 km around a point and the user only knows its position with an accuracy of 500 m. While using the primitives mentioned above, a point with a distance of 1300 m to the center of the circle would be outside the circle. This decision may be incorrect if the accuracy is taken into account.

Therefore, each continuous type may contain an accuracy attribute indicating the confidence interval of the data. Positional types contain a vertical and a horizontal accuracy, measured in meters, number ranges a float range, and times a maximal offset in seconds. The comparison rules are then modified to match if any value within the accuracy interval matches. The full schema including the quality attributes is printed in appendix A.

3.3.1.2. Composite Types

The primitive types do only permit only connected ranges and regions to simplify the matching. Sometimes, it may be desirable to match multiple regions or

inverted regions. One way would be to add further representations, but adding them for any combination would increase the complexity of the parser and the matching process. A composite type solves this by just adding two new elements. First, an OR type which may contain any number of other elements and represents the union of all ranges/regions. Furthermore, a NOT element is introduced which matches another element if its content does not match.

3.3.2. Context Manager

On the mobile client, available context needs to be managed by a central component. It has to gather the data from the sensors, notify components about changes, serialize the context data, and provide the public and the private set of attributes.

An interface to register context sensors offers methods for push and pull sensors. The former notifies the manager on any change of data, while the latter one has to be queried to get updated data. Choosing the right model for a sensor depends on the sensoring technique and the cost of acquiring the value. GPS is one prominent example of a push sensor since the data stream is constantly received from the GPS receiver. A weather forecast sensor which has to fetch data from a server should be inactive until new data is needed and thus use a pull.

Public and private attributes are separated on a sensor basis. The application may set a private, public, or blurred flag for each sensor. Blurred attributes appear in both sets, the private set with full accuracy and with a decreased accuracy in the public set.

Access to the data is possible by querying any attribute or by registration of a listener. The listener may react on any change of an attribute or only if the attribute hits a specific value. Furthermore, the application is able to get the full set of public and private attributes.

The component does also offer methods to serialize and deserialize sets of attributes as described in the representation section and to compare two sets of context attributes. The public context set is sent to the server in its serialized form in order to initiate a discovery process.

3.3.3. Service Transformation

The transformation of service descriptions is a major difference to the other approaches. Instead of using an algorithm to directly match the descriptions to the current context of the user, the attributes of the services are translated into ranges of applicable context data. Those matching rules, in the form of context attributes, are separated into two blocks. One block contains mandatory elements. These attributes have to match – if you search for a restaurant, these would be the opening times and the location. In contrast to that, the other attributes are optional and may increase the benefit of a service. The current temperature could be an optional attribute for a beer garden – while it opens on every day with a temperature of 15 degree celsius or above, it is even better with a temperature of about 20 degrees.

In case of static service descriptions, this may be computed beforehand, otherwise it has to be transformed on-demand. Any type of service description language may be used, only the previous matching logic has to be changed into a transformation. Several examples are given in chapter 6 and (Bostan, Butter, and Atkinson 2008).

3.3.4. Server-Based Filtering

After sending the request (including only the public part of the available attributes) to the service provider, the service provider uses these results to create a list of appropriate services whose mandatory rules do not contradict these attributes (van Setten, Pokraev, and Koolwaaij 2004; Hinze and Voisard 2003). The service provider notes the context attributes which are considered for the search and which influenced the order of the restaurants in the list that is sent back to the mobile device. Thanks to that these attributes contain a validity range, it is possible to re-evaluate the match without understanding the semantics of the service description by comparing the received ranges and the locally available values for each attribute.

Each context attribute which is not sent to the server may increase the size of the response since the server's ability to filter the results is reduced. While marking single binary attributes as private is expected to double the number of results, the omission of the location may result in a non-feasible increase in the

number of results if there are a lot of location-specific services available. Therefore, attributes having a big influence on the number of results and continuous values should preferably be blurred to allow a prefiltering on the server while still preserving the privacy of the user.

Depending on the size of the results, the server sends back the results or a notice indicating that the number of results is too high. In that case, hints are given to the client about which context attributes could decrease the number of hits most. These hints are calculated by counting the number of distinct values for each context attribute in the result which is not in the public set from the user. While this is no accurate estimation without knowledge about the distribution of values, it is an indication for the user which attributes could be of use in order to decrease the result size.

3.3.5. Mobile Post-Filtering & Ordering

On the client device the process is repeated, private context is considered and all services not compatible with any of them are excluded from the list. Furthermore, the list can be reordered as the correct position of the user (and not the blurred information sent to the service provider) and can be used for calculating the distances to the restaurants. An overview of the process is given in figure 3.2.

In contrast to that most services and applications that are currently offered for mobile devices today, perform the main computational effort on centralized servers (for example search engines or routing discovery). In this concept, services and applications are customized in two phases (compare also figure 3.2 as an example of a search service for restaurants). The first phase happens on the server using only the public context attributes the user wants to make available for the service provider. The service provider can use different mechanisms in order to match the context information with the service descriptions and to adapt the results to the user's context. When delivering the service descriptions or the application to the mobile device, it adds a list of context criteria which are used for matching and further context criteria which could be used if they are sent by the client. These additional criteria will be used in the next phase.

The second phase is performed on the mobile device. An application running on the mobile device scans the context criteria which are not available on the server and checks whether there are context criteria that have been declared as private

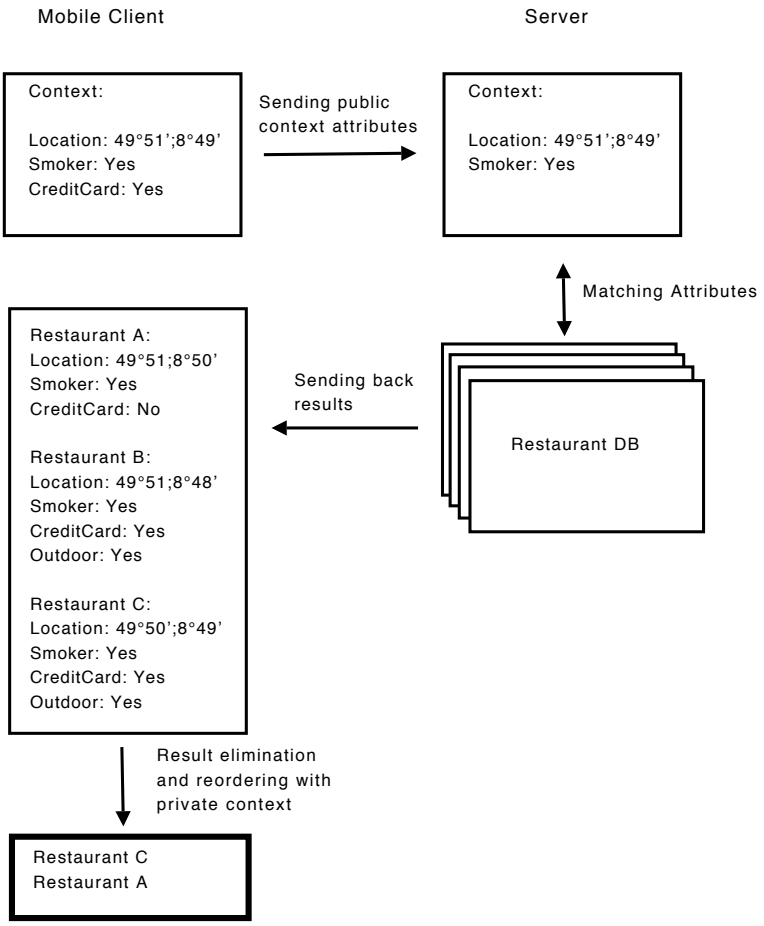

Figure 3.2.: Service Discovery

to be checked. If private context information exists, it is used for customization. A simple but illustrative example is the context information "available time". This context information can be declared as private and is only used on the mobile device for customization or narrowing of search results (of course, only if it makes sense to use this context information). Then, it is possible, amongst other things, to use the context "available time" on the mobile device in order to determine if a proposed action is feasible for the user, but it is not necessary to publish this information to the service provider. The splitted process also allows visualization of the matching process for the user.

3.3.6. Individual Importance of Context Attributes

The available information regarding the influence of context attributes on the matching process for each search result and the possibility to repeat the matching process with additional attributes on the mobile client makes it possible to re-customize the results of the search on the mobile client. This prototype implementation tries to find out which attributes that have not been considered by the service provider yet, do influence the choice of the user and how important each context attribute is for him. If the user prefers search results from the list always having some specific context attributes, the application observes this behavior, learns it, and considers the preferences of the user for future searches. The following paragraphs describe in detail how the user preferences are considered for the ranking of search results on a mobile device.

When dealing with results from search services like a restaurant search, the order of the restaurants in the list is important. The user's favorite choices are at the top of the list so that one does have to scroll or click until finding the favorite search result. However, when receiving the list of restaurants from the service provider, all private context information has not yet been used for ordering the list. Therefore, in a first step, the client discards all results from the list that can be eliminated due to existing private context attributes. In a second step, the mobile device weights the relevance of context attributes by using historical data in order to be able to better anticipate the decisions of the user. The goal of such mechanisms is to detect the relevant context characteristics and to reorder the list in such a way that the search results (restaurants) preferred by the user are at the top of the list.

When considering optional context information for resorting the list, it is unknown how strongly the different context criteria influence the user's preference and choice. Some context information (like smoking/non-smoking restaurant) may be very important for the user's choice whereas other context information (such as the distance to a restaurant) is less important. In order to consider the influence of the different context criteria on the preferences of the user, was developed an adaptive search algorithm that learns which context characteristics have been important for previous choices of the user. Since sending the decision history to the server would expose private and personal information, the whole weight calculation is performed on the mobile device.

In our approach, a weight w_i is assigned to each context information i. w_i represents the importance of context i for the user. The higher the weight w_i of a context information is, the higher the rank of those search results (e.g. restaurants) that match this context property i. For the ordering of the list of displayed search results on the mobile device, the sum $\sum_i x_i w_i$ is used. x_i denotes wether the context attribute i matches the property of the search result ($x_i = 1$) or not ($x_i = 0$). The list presented to the user is ordered in such a way that the search results with the highest sum are presented at the top of the list.

The ordering of the list offered to the user depends on the weights w_i. The goal is to determine the weights w_i so that in all previous searches the search results that have been chosen by the user would have been near the top of the list. Then, it can be assumed that for the current search the user's preferences are considered appropriately and the search result that will be chosen by the user appears at the top of the list with a high probability. For this task, an optimization algorithm is necessary which determines the weights w_i so that the search results (e.g. restaurants) that are preferred by the user are on the top positions of the list. Such an optimization algorithm can utilize the previous user's choices for the evaluation of different combinations of w_i.

When the user executes a search service for the first time, there is no information available regarding the importance of the different context characteristics. Therefore, when using a search service for the first time, we assume that all context characteristics do have the same importance and the list of results that is received from the service provider is reordered in such a way that all context criteria are considered equally. For example, for a non-smoker who does not want

to use a car to get to a restaurant, a non-smoking restaurant is as attractive as a restaurant where he does not need a car.

3.4. Prototypical Implementation of the Service Discovery Concept

The components concerned with service discovery have been implemented to show their utility, quality, and feasibility according to guideline three in 1.3. Moreover, it serves as a test environment for the creation of service and the other components. Some challenging parts and their design choices are outlined within the next sections. The components are used for the sample applications in chapter 6.

3.4.1. Server Component

The prototypical test server contains a storage for XML service descriptions that may contain context rules. Alternatively, an extendable style sheet transformation language (XSLT) document may be provided to be able to generate the context attributes out of the service description. It is applied by the server before the discovery process.

A more sophisticated server following this concept and using a scalable XML Bostan et al. database and model-based transformations was developed during the course of the SALSA project by Bostan et al. (Bostan, Butter, and Atkinson 2008).

3.4.2. Context Manager

In order to test the behavior and generation of different context sets a context management component with several context sensors was developed. The distinction between private, blurred, and public attributes can be done using a user interface giving an overview of all sensed data.

3.4.3. Learning

For the optimization algorithm that should find the optimal w_i, tight technical restrictions are imposed by the limited capabilities of mobile devices. The most important are:

- The algorithm must have low heap memory usage and small code size.
- It must be possible to partition the algorithm in very short execution intervals to not interfere with user tasks.
- It should be possible to use old weights w_i as a starting solution.
- The algorithm should not rely on floating point arithmetic since these may be slow on mobile devices.

To find the optimal w_i, linear optimization methods cannot be used as the quality of a solution (a solution can be described by a parameter setting for the w_i) is non-linear and depends on the user's choices. To solve the problem of finding the optimal values of w_i, a a genetic algorithm (Goldberg 1989) can be used. This type of optimization algorithm is able solve non-linear problems, it can work without any noticeable drawbacks for the user, it is possible to implement it with very few lines of code, it is possible to stop the execution after every single fitness evaluation without the need to hold extensive state information in heap, and meaningful results are available anytime. Other optimization techniques like simulated annealing would probably lead to similar results but require floating point numbers and more (pseudo) randomly generated bits. These operations are especially expensive on todays mobile devices.

The implementation the genetic algorithm for the weight optimization on a mobile device in a Java class file fits in about 5kB. Such a small application can easily be held in memory all the time. The history (previous user choices) is read in sequential access for every fitness evaluation (one fitness evaluation calculates the quality of different w_i using the history where a top-ranked service means high fitness and a low-ranked low fitness) which allows efficient loading for every storage medium. The algorithm gets some processor time for fitness evaluations during short idle intervals like the time waiting for a reply over the network. Using such time slots does not affect the battery lifetime so much, because an active network connection does not allow a mobile device to use a battery saving

state. Furthermore, in such time intervals, the user has to wait for the reply so that it will not it slow down the user interaction.

3.5. Transferability of the Concept to Navigation

Routing is an important service for mobile applications. The mobility of the user leads directly to the need of routing directions. It is similar to service discovery which need large amounts of up-to-date data, such as maps and traffic patterns, which change over time. Current navigation solutions are using static maps on a local storage or let the server process all route calculations. Therefore, users are having either old and incomplete data or giving up their location privacy. Sending out the location of the user for routing purpose has the potential to eliminate the privacy enhancing techniques shown in this thesis and so a privacy-aware navigation solution is needed. This section shows how the concepts of the distributed service discovery presented in the previous sections are also applicable to routing. First, the concept which splits the processing between the mobile device and the server will be explained. Afterwards, some measurements are done with a prototypical implementation to find suitable parameters for the trade-off between privacy, used bandwidth, and computational cost.

3.5.1. Introduction to the Problem Domain

With an increased usage of context-aware services, automatic routing between visited points and used services does also increase. Disclosing the start and the destination of a route may be used to infer the user's next service. The benefits of the privacy-aware service discovery will therefore be destroyed. To have all maps locally on the mobile device, decreases the accuracy since maps and street conditions often do change and current traffic data cannot be used. Furthermore, calculating routes with the map data of a whole country is computationally expensive and, therefore, reduces the battery lifetime.

A routing algorithm is needed which finds the optimal route with minimal data on the mobile client and without giving accurate location data to the server. "Optimal" may be the route with the shortest traveling time or shortest distance. In this section, the shortest traveling time is selected as the optimization criteria. The shortest distance is just a special case with an assumed constant speed on all streets.

3.5.2. Conceptual Elements

Similar to the discovery approach, the routing is distributed over a server and the mobile client application. As already mentioned, the user's current location and travel destination is sent to the server with decreased accuracy. The server therefore only knows where the client is located and the desired destination within a circle around a point with a given radius r_1 and not the exact location. It is then possible to show the route by downloads the map data for these two areas and the all connecting routes between these areas which are part of optimal routes between any point in the origin and destination circles . Connecting routes are defined as the unique route segments outside the areas which connect every pairwise combination of points within the two areas. Points near the edge of each circle have many exit routes out of the circle leading to many connecting routes. The central point in following algorithm is to reduce these connecting routes to allow a bandwidth and computation efficient communication between the server and the client.

A streetmap can be represented as a graph where each street/road represents a vertex between two nodes, which in turn represent an intersection, point of interest, or dead end. The cost associated with each vertex is the length or the time to travel that road. Routes for cars are usually calculated using the "fastest" route from start to destination. Therefore, the length of each vertex, is multiplied with the reciprocal value of the average speed assumed for this type of road.[1] Since the shortest route is just a special case of the fastest route with the assumption of equally fast travel on any road only the fastest route case is examined.

While the number of connecting routes is high and grows with r_1, there are only a few distinct middle segments within those routes. In typical street-maps, this approach assumes many routes between nearby starting points to the same (or similar) destination join each other within a small distance of the starting points. Therefore a second region with an increased size around the middle of the departure and destination region is introduced.The client then gets the map data for the larger region (size r_2) while only starting/end points within the

[1] If an A*-search (Hart, Nilsson, and Raphael 1968) is used to find the optimal route the fastest road needs to have a factor of 1 and each slower road a bigger one. The algorithm guarantees to find an optimal solution if $h(n)$ (the linear distance) is the shortest possible path only.

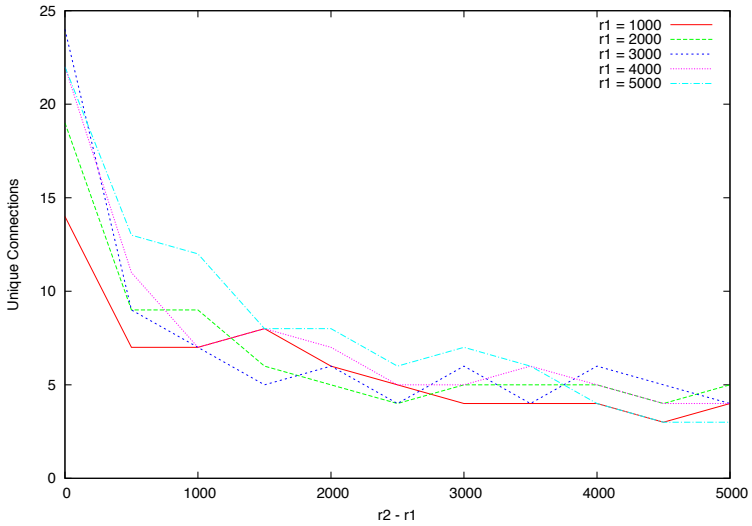

Figure 3.3.: Unique Connections w.r.t. Δr

smaller region are used to find connecting routes. The expanded area is then used to eliminate duplicate connecting routes. Using only starting points within the inner area, decreases the number of distinct connecting routes significantly. Therefore, only a few routes need to be known outside the area where the start of the route is. This assumption is based on the fact that routes with a smaller cost are favored and long roads are typically the faster ones (e.g. motorways). All segments within every optimal route between two points in the start and destination areas are included, so the optimal route can always be computed using the maps of the areas and the connecting routes.

To test the assumptions, a simulation was performed. The simulation setup is explained in section 3.5.4. 500 random points in a circle with radius r_1 were chosen. The simulation takes place in the city of Karlsruhe, Germany. For each r_1, several Δr which represent the difference between the inner and the expanded circle are used and the number of different routes to a given destination starting outside the expanded circle are counted. Figure 3.3 shows the effect of increasing Δr on the number of unique connections.

Most routes between the two circles tend to join each other into some main routes, because bigger roads do have a higher average speed and are therefore favored over other roads. To fully exploit those joined paths which result in a lower number of distinct connecting routes, the mobile nodes get a bigger part of the map by increasing the radius of the circles.

The route calculation is distributed between the server and the client. The server sends all routes between the expanded areas and the client calculates the remaining parts of the routing. Since those graphs are much smaller than graphs representing routes within whole countries, the route calculation costs on the client also do decrease.

3.5.3. Finding Connecting Routes

The server uses random points in the inner circles and calculates all routes between all pairs of nodes in circles A and B with radius r_1. The number of routes is $|A| \times |B|$ (with $|X|$ the number of random points in X). The routes needed on the client side are the unique routes starting with the last point within the expanded areas with r_2. A large number of nodes in dense areas may lead to a very high number of calculated routes and prohibitive in terms of computation cost.

Depending on the size difference of the radius, the number of unique routes does not increase significantly with a higher number of random points n beginning with $n*$. To find suitable values for Δr and $n*$, several simulations examining the computation/accuracy tradeoff were performed.

3.5.4. Simulation Setup and Results

To get realistic simulation results, real map data is needed. Therefore, the streetmap data of the OpenStreetMap project[2] has been used. The map data is collected by volunteers and thus city maps are not always complete.

The simulation uses a subset of the OpenStreetMap data exported in the first week of March 2008. In the subset, each node and way within the administrative borders of Germany are contained in a 2GB XML File. To improve the performance of the route calculation, the file is parsed once and the nodes are stored within a hash map. The ways are stored as pairs of nodes and the distance between the nodes is precalculated. The city of Karlsruhe, Germany, and the city center of Mannheim, Germany, are esteemed as quite complete by the project and thus used Karlsruhe and Mannheim was used as places for our simulation (see figure 3.4). The results were also verified using random rural areas and some

[2]See http://www.openstreetmap.org/ for details

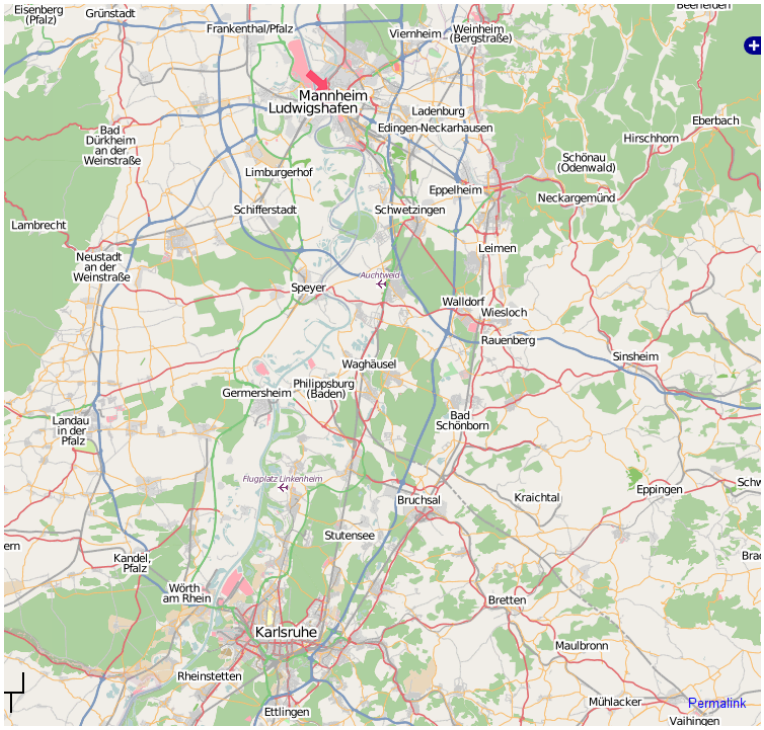

Figure 3.4.: Region used in Simulation (source OpenStreetMap)

British city maps, but using German speed limits. The rural tests did all show less routes and a smaller number of nodes. British cities were similar to the German ones. The simulation results do strongly indicate that the two German cities are good representatives and that rural areas generally perform even better than urban ones, even though the British speed limits were not used.

In the figures, the increase of map data size for an increased r and the number of unique external routes with an increased Δr is shown. A graphical representation of the areas with colored connecting routes can be seen in figure 3.5. The shaded areas are not relevant for the routing and therefore are not transmitted.

In figure 3.6, the number of nodes in a circle with the radius r_1 with its center located at the main station in Karlsruhe is shown.

Figure 3.7 shows the size of the map data for these circles in KB. An average number of connections of 1.95 per node and an average of 10 bytes of compressed street names for each connection is assumed. These values were extracted out of the German map with all nodes and connections relevant for car navigation.

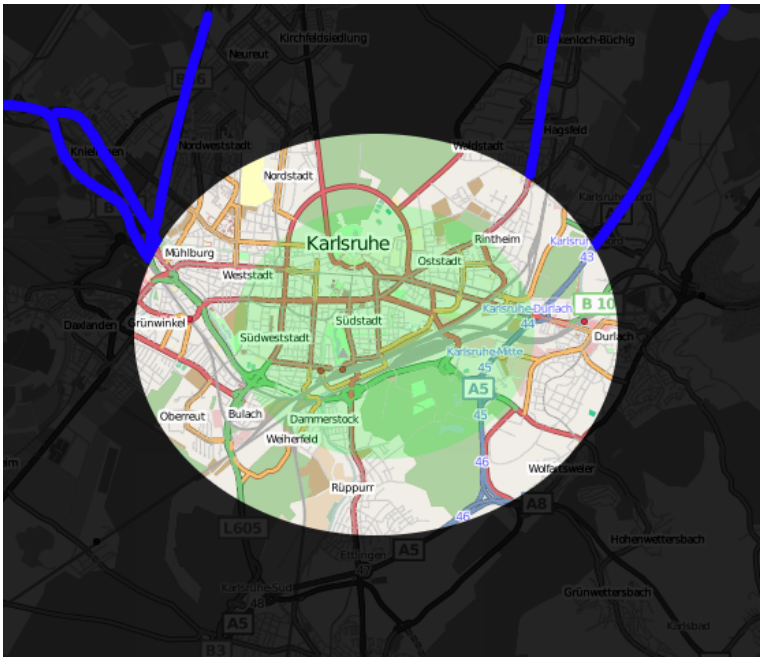

Figure 3.5.: Karlsruhe with Highlighted Areas R_1, R_2, and Connecting Routes

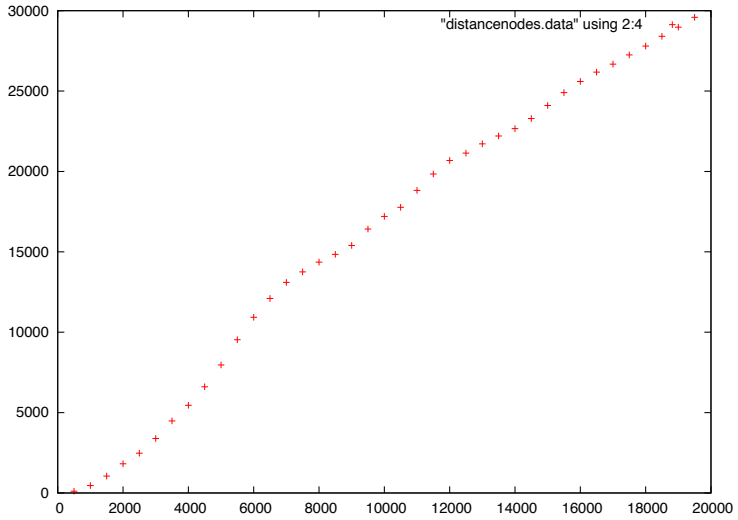

Figure 3.6.: Number of Nodes w.r.t. r

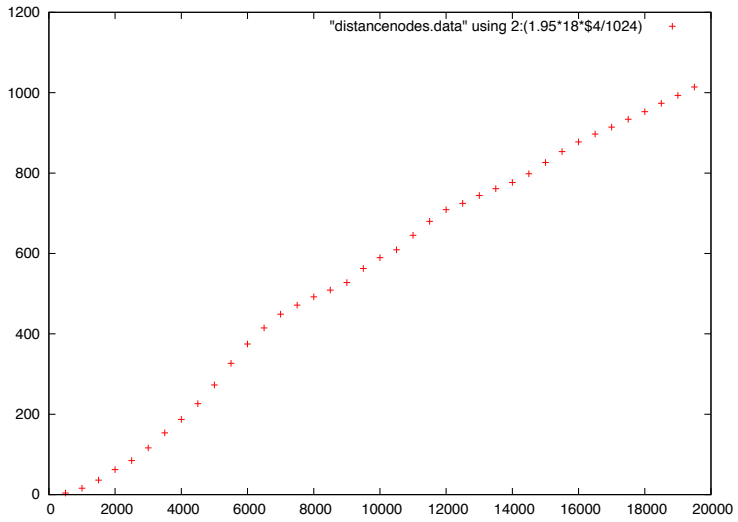

Figure 3.7.: Size of Mapdata of Nodes w.r.t. r

Road Type	Factor	Avg. Speed
motorway_link	1.2	100
motorway	1	120
trunk_link	1.4	86
trunk	1.2	100
primary_link	1.4	86
primary	1.3	92
secondary	1.7	70.0
tertiary	2.1	50
residential	4	30

Table 3.2.: Avg. Speed per Road Type

The smaller size of the data does not only decrease the data transfer volume, but also the complexity of route calculation[3]. Furthermore, the memory usage is drastically decreased, which is also important in mobile applications as outlined in the requirements section.

3.6. Evaluation of the Service Discovery Concept

Using context information for the customization of mobile services is a promising direction to increase the usage of mobile devices. However, when dealing with context information, privacy issues are very important. Therefore, mechanisms are necessary that enable the user to control the usage of context. This approach distinguishes between private and public context information. Public context information can be used by service providers for the customization of mobile services. Private context information can be used on the mobile device for further adaption of mobile services with respect to the user's context.

It is illustrated for search/discovery services how context information can be split into public and private context. In search services, a mobile user receives a list of search results from a service provider which can be sorted according to the user's context. The goal is to sort the list in such a way that the items chosen by the user are at the top of the list. We propose an optimization approach that orders the results delivered from the service provider with respect to the user's private context information and to previous choices of the user. This approach requires only minimal computational effort and makes optimal use of the limited resources of mobile devices.

Summarized, the requirements established above are fulfilled as follows. The functional capabilites of the component within application is shown in chapter 6.

Extendable. Any context attribute can be expressed as one of the primitive types or a combination of several others. In contrast to other approaches, new behavior depending on a value of a context attribute does only have to be implemented on the server side and the mobile client is able to refine the search using the rules sent by the server without any extension on the client side.

[3]While the complexity of A* does only depends on the length of the shortest route, this is only true for an optimal $h*$ which is not given in the case of different road types.

Reusable. Being part of a framework which should allow creation of arbitrary context-aware applications, it has to be reusable together with a partial concrete implementation of the problem solution (Wirfs-Brock and Johnson 1990). In the domain of a context-aware service discovery, this means that it should be usable to discover different kinds of services within varying application domains. It is therefore linked to the first requirement of extendability.

Privacy preserving. Using the aforementioned service discovery, the user has full control over the data which may be sent to the service provider. A critical point is the increase in used bandwidth for every unset context attribute. The approach is not feasible if every attribute is declared private and has best results having the continuous attributes only blurred instead denying the transmission completely. To improve the situation for the user, it could be combined with other anonymization approaches.

Learning from previous user behavior. Matching of the optional context rules is done on the mobile applications. Therefore, it knows about the rules used to rank the service and may use the user behavior to improve the ranking for future searches.

4. User Interfaces

In this chapter, an adaptable user interface component for the context-aware framework outlined in the previous chapter is introduced. It starts with the first section detailing the requirements of mobile context-aware user interfaces based on current scientific literature, followed by an overview of existing approaches for desktop and mobile systems. Then, a new concept based on the extendable user interface language (XUL), cascading stylesheets (CSS), and context rule-based presentation changes is shown. A prototypical implementation for multiple configurations and devices is described in the next section. The chapter ends with an evaluation of the prototype against the outlined requirements.

4.1. Challenges of User Interfaces on Mobile Devices

In a literature review, the following requirements and challenges in addition to the general requirements outlined in chapter 2 were identified and will be further elaborated in the following sections. A general overview of mobile UI requirements can also be found in (Satyanarayanan 1996):

- Automatic adaptation for heterogenous devices (see 4.1).
- Adaptation to context (see 4.1).
- Usage of private context (see 4.1).
- Suitability for low processing power of mobile devices (see 4.1).
- Customizable by the user (see 4.1).
- Simplified application development (see 4.1).

Heterogenous Devices Mobile devices available in the market today are heterogenous with regards to their input and output capabilities, using different platforms and featuring a multitude of different display resolutions and display orientations (i.e. landscape or potrait mode)(Lee, Ko, and Fox 2003). It is often necessary to manually adapt an application to multiple devices at development time taking into account the different display and input capabilities. The number of available platform configurations, such as J2ME CDC(Sun 2005) or CLDC(Sun 2004), further increases the number of different implementations needed for a portable mobile application.

Table 4.1 shows the varying screen size of some current and popular mobile phones and PDA (Admob 2008). The table 4.2 shows which application platforms are available for the predominant operating systems. Today, most applications need to be customized manually for each programming platform and screen size. Hundreds of different product versions are common for popular applications like games to cover a wide range of devices on the market. A UI framework should be able to cover as many screen resolutions and platforms as possible to reduce the development effort for context aware applications.

Device	Screen Size	Resolution	Colors	Orientation	Keyboard
Apple iPhone	3.5 inch	480x320	16m	p. or l.	on-screen
Nokia N95	2.6 inch	240x320	65k	portrait	number / T9
Nokia N810	4.13 inch	800x480	65k	landscape	full-qwerty
Nokia 2310	1.2 inch	96x68	65k	portrait	number / T9
MDA Vario II	N/A	240x320	65k	landscape[1]	qwerty

Table 4.1.: Example Device Screen Properties
Example Device Screen Properties (source: manufacturer websites)

Operating System	Manufacturer	Platforms
Windows Mobile	Microsoft, QTek, ...	.NET, Java CDC & CLDC[2]
Symbian9	Nokia, SonyEricsson, ...	Symbian, Java CLDC
Symbian8	Nokia, SonyEricsson	Symbian, Java CDC
Linux	various	mostly J2SE compatible
iPhone (OSX)	Apple	none available
Other Phones	various	most are CLDC
Android	HTC, Google	custom Java configuration

Table 4.2.: Operating Systems
Operating Systems (source: manufacturer websites)

Adaptation to Context Readjusting the user interface to the user's current situation to better suit his needs and desires does improves the usability (Dey 2001). Kaasinen states: "An efficient way of improving the usability of mobile services and applications is to adapt the contents and presentation of the service to each individual user and his/her current context of use. In this way, the amount of user interaction will be minimised: the user has quick access to the information or service that (s)he needs in his/her current context of use. The information can even be provided to the user automatically." (Kaasinen 2003).

Integrating the presentation readjustments with the application logic builds a huge burden for application developers. A UI framework should eventually allow easy adaptation of the user interface to the current context of the user without increasing the development work.

Using private context The notion of private context was introduced in the foundation chapter. While varying projects use a different notion, most of them do have some separation between attributes available to random service providers and those only to the user for privacy reasons.

"In our group interviews, people were worried about their privacy and the 'big brother' phenomenon when considering services enabling people to be located" (Kaasinen 2003). As stated in section 2.3, privacy is an important aspect for mobile context-aware applications. Therefore, a user interface can only adapt to context data the user is not worried about or has to be separated from the application logic to make sure that the private information does not leave the mobile device. In a scenario with downloadable, untrusted components, the pre-installed user interface component may still be seen as trusted.

Low Computational Costs While mobile processor speeds increased dramatically in recent years, memory performance is still slow and only minimal if any instruction or data caches exist. So, mobile processors are still slower than common desktop CPUs some years ago. Therefore, the solution may not slow down the execution of downloaded components a lot. Especially the start-up delay imposed by UI instantiation needs to be small (Varshney, Vetter, and Kalakota 2000). Tolia et al. states that the "crispness" of interactive response is the most critical user experience performance measure. Here crispness is defined as the time between an action and a visual reaction by the application (Tolia, Ander-

sen, and Satyanarayanan 2006). The user expects so see immediate results on a "mouse click" or other interaction.

Personalization Customizing the applications according to their user's explicitly stated (in contrast to infered) preferences is called personalization (Lee and Benbasat 2003; Rayport and Jaworski 2001). According to Bauer et al. personalization is an important factor for the adoption of mobile services by consumers (Bauer, Reichardt, Exler, and Tranka 2007). Customization of mobile devices through so-called wallpapers, ringtones, or pictures is a multi-billion dollar business in Europe and Japan (May and Hearn 2005) and an important lifestyle component of mobile commerce.

Development Effort One of the most important challenges in UI creation for mobile devices is the development effort. Currently, most user interfaces have to be developed for different devices many times (Eisenstein, Vanderdonckt, and Puerta 2001) and are a significant cost factor in development of mobile applications.

4.2. Existing Mobile UI Approaches

Several user interface frameworks for different platforms do exist. While most of them are only available on desktop PCs, some of them are available for mobile devices. For context-aware user interface frameworks, more requirements need to be addressed. In the next sections, a brief overview of some exemplary UI frameworks and outline the requirements for context-aware user interfaces is given. If the concept is available on different platforms the Java version is shown here for easier comparison with the prototype. Most concepts exist in similar a form for most other mainstream platforms.

4.2.1. Model-based Development

Model-based approaches use a model of the user interaction to generate user interfaces for different platforms (Szekely, Luo, and Neches 1993). Eisenstein introduced an abstraction for describing user interfaces with models. The models

are often designed using a graphical environment where interface elements are semantically connected to each other and to event. These are then transformed to platform dependent models with varying display sizes or usage contexts (Eisenstein, Vanderdonckt, and Puerta 2001).

4.2.2. Server-side Transformations

Using server-side transformations, a device independent description of the interface does only exist on the server. Depending on the requesting device, the description is transformed into a device dependent description or program which is then transferred and executed. If the device independent description is a model, the transformation may be a special case of a model-based approach where the platform dependent model creation is deferred. Capabilities for the target device need to be stored on the server or transferred with each request.

The XML User Interface Language (XUL) is an XML-based markup language for the description of user interfaces. It is mainly used by the Mozilla Foundation in products like the Firefox browser or Thunderbird mail client, but is also increasingly popular in the area of Web applications. For desktop applications, Java XUL rendering components exist which allow the usage of XUL for Java applications. XUL is used by several other projects for mobile devices (Ye and Herbert 2004) to specify user interfaces and then transform the XUL to device specific HTML or WAP pages. Some other projects allow rendering XUL or alternative transformation to HTML depending on the device using mobile agents (Mitrovic and Mena 2002).

Some approaches for mobile UIs use XUL and transform it into HTML. Some use a Swing XUL renderer on high end mobile devices like laptops (Mitrovic and Mena 2002; Kao, Shen, Yuan, and Cheng 2003). In that case, JavaScript logic may or may not be inherited for the transformed content. Other approaches use XUL as an origin for other transformation or compilation into Java bytecode.

4.2.3. Java Frameworks

The first UI toolkit for the Java platform was the Abstract Window Toolkit (AWT) included in the first version of Java. It features abstraction from native platform GUI elements in Java to offer cross-platform compatibility. Even though

the AWT programming model changed a bit with Java 1.2, it is not the preferred framework on the J2SE anymore. However it is available with every version of J2SE and even with J2ME CDC Personal Profile and thus is a good foundation to reach many platforms with a single implementation. In our prototype implementation, it was decided to base one of our back ends on AWT.

Swing, a newer core Java UI framework, was introduced in Java 1.2. It is implemented as pure Java components building upon the native bindings of AWT. Swing features tremendeous easier programming and much easier integration of own components while offering different look and feels for different platforms to better integrate into each host platform. However, Swing needs considerably more system resources, so it is not suited for today's small mobile devices, but laptops or tablet PCs. (Traynor 2008).

4.2.4. Mobile Information Device Profile

The Mobile Information Device Profile (MIDP) (Sun 2004) was introduced for the CLDC platform on very small devices like mobile phones or low-end PDAs, but it is currently also the most widely deployed platform even on high-end PDAs. The *javax.microedition.lcdui* packages only allow very small set of widgets to be used and the look of these widgets cannot be influenced by the application. In that framework, the screen is always occupied by elements of the *Displayable* type which is in most cases a *Screen* embedding a *Form* which further embeds some standard widgets. In addition to those standard widgets, a *Screen* may be replaced by a *Canvas* which allows custom drawing and reactions to key events. Often both ways are combined to get the enhanced graphic capabilities of a *Canvas* together with the handling of phone input methods like T9 in a *TextBox*. (Muchow 2001)

4.2.5. HTML / Javascript

An increasing number of applications for mobile devices use server side processing and only WAP, HTML, or XHTML browsers for the user interface. Often, there are different versions of each HTML page adapted to different screen resolutions. Many mobile devices do only offer a very limited subset of Cascading Style Sheets (CSS) for styling HTML content and often JavaScript is not supported, thus all

actions are server-based which leads to long response times, because of the high round-trip times in mobile networks. (Nichols, Hua, and Barton 2008)

4.2.6. Android

Android is a new open source operating system for mobile devices introduced by google in late 2007[3]. It is said to include an XML definition of user interfaces which adapt similarly to MIDP to different screen sizes and devices.(Köchy 2008)

4.2.7. Evaluation of Existing Approaches

In table 4.3, the mentioned approaches are evaluated according to the requirements elaborated in section 4.1. While most approaches are very good in one or two categories, none of them fulfills all requirements stated above.

Model-based and server-side transformations are similar with respect to their start with a device independent model and the adaptation of content prior delivery to the mobile device. Those can be quite fast since they have no need for adjustments on the device and all unnecessary descriptions may be left out. While this is an advantage for performance, it means that the server needs to know something about the used device and every context that should be used for the adaptation.

MIDP does not allow any customization of the look and feel of the widgets, but does work seamless on every device. The applications will look different depending on the used Java implementation and nothing about the look can be said beforehand. The AWT is designed for a multi-window enviroment and not suitable for small devices. Both are not adaptable to the usage context.

Using a well-designed HTML page, it would be possible to design pages suitable for a wide range of devices. Implementing the application logic in JavaScript involves expensive processing of DOM operations and redraws. Furthermore, the high latency of mobile networks leads to long page loading times since HTML application logic is page based. Context-usage is not envisioned in HTML.

The table shows the summarized results for the different approaches. All of them are exceptionally good in some respects, but none fulfills all requirements

[3]The description of Android is based on release m5_rc14 of the SDK which is the latest available version on 24.2.2008.

at all. Especially the combination of adaptation and privacy is not available anywhere. The next sections show a new proposal to combine these features in one framework.

Project	Heterogenous Devices	Adaptation to Context	Using Private Context	Low Processing Power	Customizable	Development Effort
Model-based	+	-	-	+	-	+
Server-side Transformation	+	-	-	+	-	+
AWT/Swing	-	-	-	-	+	-
MIDP	+	-	-	+	-	-
HTML / JavaScript	+	-	-	-	-	++
Android	+	-	-	+	-	+

Table 4.3.: Evaluation of Related Work

4.3. User Interface Concept

In this section, a component will be introduced which enables the creation of applications for different mobile devices and fosters a GUI which reacts to context changes without having the need of any application logic supporting it. We propose an extension to XUL in order to enable the easy creation of applications which adapt themselves to different devices and user contexts. The Java platform was chosen for the prototype since it is available for most mobile phones and PDAs. In section 4.3.2, a prototypical implementation for the J2ME Profiles CDC and CLDC, including a compilation approach to speed up UI creation, is presented.

The usage of sensitive context data restricts the processing to the client side since no sensitive data should ever leave the users mobile device. While techniques like anonymization (Tatli, Stegemann, and Lucks 2005) may be suitable to protect the privacy for some data, it is desirable to use all data. For example, the real name of the user or the home address for a personalized address would destroy the benefit of anonymization. Therefore, all customization has to be done on the mobile device which extends the restrictions imposed by the quite low processing power of mobile devices.

Customization must be done for the current context as well as the current device capabilities, such as screen size. As outlined in chapter 2, according to the context definition of Dey (Dey 2001), device characteristics are a part of the user context. Therefore, the adaption to different devices is a special case of a flexible context-sensitive user interface adaption which simplifies the whole concept and improves usage from a developer perspective. The context representation shown in chapter 3 handles screen orientation and size in the same way as all other context data, paving the way to a unified usage of device properties and user context.

It is generally seen as an important concept in UI toolkits to separate logic, semantic models, and the styling from each other (Lopes and Hursch 1995). HTML and CSS are one example for the usage of this paradigm. While inline Javascript and inline styles in today's AJAX applications blurr this distinction, the original goal was to put the semantics of the representation into the HTML, the styling into the CSS file, and the data/logic on a server side. Following this approach, XUL as the language to model the semantic description of the UI

and CSS to style it was chosen. XUL as user interface description shows some promising results in other projects like the Mozilla Firefox browser which has its whole user interface designed using XUL.

The XML User Interface Language (XUL) (XUL Tutorial 2006), introduced by the Mozilla Foundation, is an XML based standard for describing user interfaces. It can be mixed with many other markup languages supported by the Mozilla or Firefox browser like HTML, MathML, or SVG. On desktops, the application logic is programmed in JavaScript which accesses the XUL via a DOM interface.

A XUL renderer was implemented for small mobile devices. For styling, a subset of CSS was used. Parsing XUL turned out as too computationally expensive on small devices for a reasonable fast rendering, so a compilation approach was considered (section 4.4.7).

An XUL example could look as follows:

```
<?xml version="1.0"?>
<window id="window" orient="horizontal">
        <vbox id="toplevel">
                <image id="startimage"
                        src="startscreen.jpg"/>
                <label id="greeting" value="Welcome" />
                <button id="setup"
                label="Customize profile"
                oncommand="gotoProfile"/>
        </vbox>
</window>
```

In this example, a window would be created consisting of a vertical box with an image and a button. To compensate for the smaller display sizes, a single window model is used and, therefore, each XUL file defines a single window which is exclusively visible to the user. The styling of the individual elements is done using CSS level 3, whose level of support may depend on the type of device.

The application logic should be separated from the user interface. Since parsing a scripting language is quite expensive (see measurents in the evaluation), the direct usage of the Java language as programming language for the application logic was chosen. Maintaining the DOM, providing all necessary operations

is quite memory intensive. Therefore, an API to access the XUL components programmatically without a full DOM interface was developed.

In this case, an XUL renderer for different Java configurations / platforms and control the user interface via plain Java classes, allowing source compatibility of application and UI logic between J2SE AWT, J2ME CDC Personal Profile, and J2ME MIDP, as well as the use of context information in an application transparent way as outlined in the next sections was developed.

4.3.1. Adaptation to User Context

Bringing in the definition from the foundation chapter, according to Dey "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." (Dey 2001).

Here context is seen as anything that may be used to adapt a user interface to the user's current needs and situation. As the user interface does also need to be adapted to the device capabilities, the device capabilities can be seen as context attributes, just as the user's current position or activity. Thus, our UI framework does only have to adapt itself according to context changes or the currently available context information at instantiation of the UI.

While the structure of the user interface is specified using XUL, the appearance is specified using Cascading Style Sheets (CSS). To better match the current context, XUL and CSS were extended to allow loading a CSS depending on a pre-specified user context. For globally used style sheets, the standard HTML notation may be used:

```
<link rel="stylesheet" type="text/css"
  href="standard.css">
```

or

```
<style type="text/css">
        h1 {color: red}
</style>
```

Another tag is specified to add or remove style sheets depending on the current context. The *contextstyle* element adds a context constraint using a serialized

form of context data in a format defined in (Butter, Deibert, and Rothlauf 2006) and the style tag. Using this tag, it is possible to add styles depending on the user's context and thus reaching some dynamic behavior of the UI. Since it is possible to hide elements using CSS attributes, interface elements may even be hidden on small devices or in some contexts. An example using a bigger font for headlines in case the user is walking would look like this:

```
<contextstyle>
        <style>
                h1 {font-size: +1};
        </style>
        <String name="Activity">
                <Value>Walking</Value>
        </String>
</contextstyle>
```

Alternatively, larger style sheets may be included using the *src* attribute for the style tag:

```
<style src="biggerfont.css" />
```

The selection of relevant style sheets is done dynamically, so the layout and appearance of the UI changes automatically if a user starts to walk and the context management recognizes the change. This can be done without any interaction of the application logic.

4.3.2. XUL on mobile devices

Mobile devices often do not feature a lot of screen space and are memory constrained. A UI framework which has to work on most mobile devices needs to take these constraints into account. Therefore, it was chosen to not support JavaScript scripting and not implement a full DOM interface. Furthermore, not all CSS properties and XUL elements are implemented. Nevertheless, a subset of XUL and CSS can be used which allows the creation of applications rich of features. While many visual CSS properties are supported on the Personal Profile version, most are ignored in the MIDP version. Since the Personal Profile AWT is a subset of the AWT available in the Java Standard edition, the Personal Profile does also work on desktop computers. In all cases, the same source can be

used on both platforms and the framework automatically decides which backend to use.

4.3.3. Multi-Screen Dialogs

In some scenarios, multi-screen dialogs are desirable. These are well known for initial settings dialog, called "wizards" on Microsoft Windows or "druids" on Linux systems. Those screens are strongly connected and typically no data processing or application logic is involved between the screens. To foster easy creation of these wizards, a language for their control was developed.

For each XUL window using this feature, a ".rul" file exists. How to use this feature for rapid prototyping of mobile applications is described in section 6.2.

4.3.3.1. Script Syntax

The ".xul" files consist of the following sections:

- **Comments** start with a hash-sign and will be ignored.
- **method** sections begin with the name of the method followed by a colon. Then, this method is used as an event handler in the XUL. The following expressions are executed.
- **+start** denotes a special method name which is always executed on loading.
- **+contextchange** is another special method, followed by some context rules. Whenever these results of the rules come true, the corresponding expressions are executed.

Expressions may be variables, assignments, and commands. Variables are without type (i.e. may be of any type and are transparently converted) and each element of the XUL file may be used as a variable. Whenever a variable is used which does not appear in the XUL, a dummy node with that value is inserted. Those are globally valid in each following XUL file. An assignment is denoted by a "=" sign and the left variable's value becomes the value of the right variable.

The only predefined commands are "changeScreen()" and "popup()". The former opens the window denoted by the file name within the brackets together with the .rul file. The latter opens a popup to inform the user about an error.

Putting everything together, a multi-screen wizard could be implemented like this:

```
#comment
+oncontextchange:
    <context>
      <String name="Room">
        <Value>Bedroom</Value>
      </String>
    </context>
    changeScreen("bedroom") # changes current
                            # screen to "bedroom.xul"
                            # and uses the
                            # "bedroom.rul" rules

methodname1:
    changeScreen(other)

methodname2:
    popup("Hello World")
    var1 = tagID.value
```

4.4. Prototypical Implementation of the Concept

The XUL implementation has to account for the variety of target devices and thus has to require only small amounts of memory and processing power. Therefore, a full DOM of the XUL would be too memory consuming and it has been decided to implement an easier and more lightweight API to access the XUL widgets.

4.4.1. API

The UI component exposes an API allowing the creation of windows using an XUL file and modification of its elements. Similar methods are available to those found in W3C DOM, but they are restricted to some simpler variants in order to save memory. The API enables the component to reuse the parsing tree

directly for rendering of the elements. Some selected methods are described in the following sections:

- *XULToolkit.parseXUL(String xul, Object handler)* is a static method in XULToolkit. It gets a *String* containing the XUL and an eventhandler as argument, returning a *XULWidget* which later may be shown as a window.
- *XULToolkit.createElement(String tagname)* is a static method which creates a corresponding new element using the given tagname to determine its type. The returned object is also of the *XULWidget* type which can be added to other Widgets as a child.
- *XULWidget.setAttribute()/setAttribute()* sets or gets attributes of an element. The attributes correspond to the XML attributes but may also contain other *Object*s instead *String*s to foster easy storage of data within a UI element.
- *XULWidget XULWidget.findByID(String id)* searches for children with the given id. This can for instance be used to find a label in order to change its text.
- *XULWidget.getChild(int i)* returns the child "*i*" of an element. Iterating over all children can be achieved using this method.
- *XULWidget.addChild(XULWidget)* adds a "child" to a node.

Additionally, several convenience methods for common do tasks exist. One example is the creation of a *ListItem* which should be placed within a *ListBox*. Instead of creating a new element, setting the value and adding to the parent, the method *ListBox.appendItem(String)* exists.

Events .

All events from the DOM Event Specification (W3C 2005) concerning mouse and keyboard events are supported (e.g. *onclick, onmousedown, onkeypress*). The event handler is the name of a method in the specified handler which is then called with the target XULWidget as argument. It was chosen to handle invocation for the eventhandler via reflection instead of a single handle(String event, XULWidget) method. While the latter would be possible and easier to port to

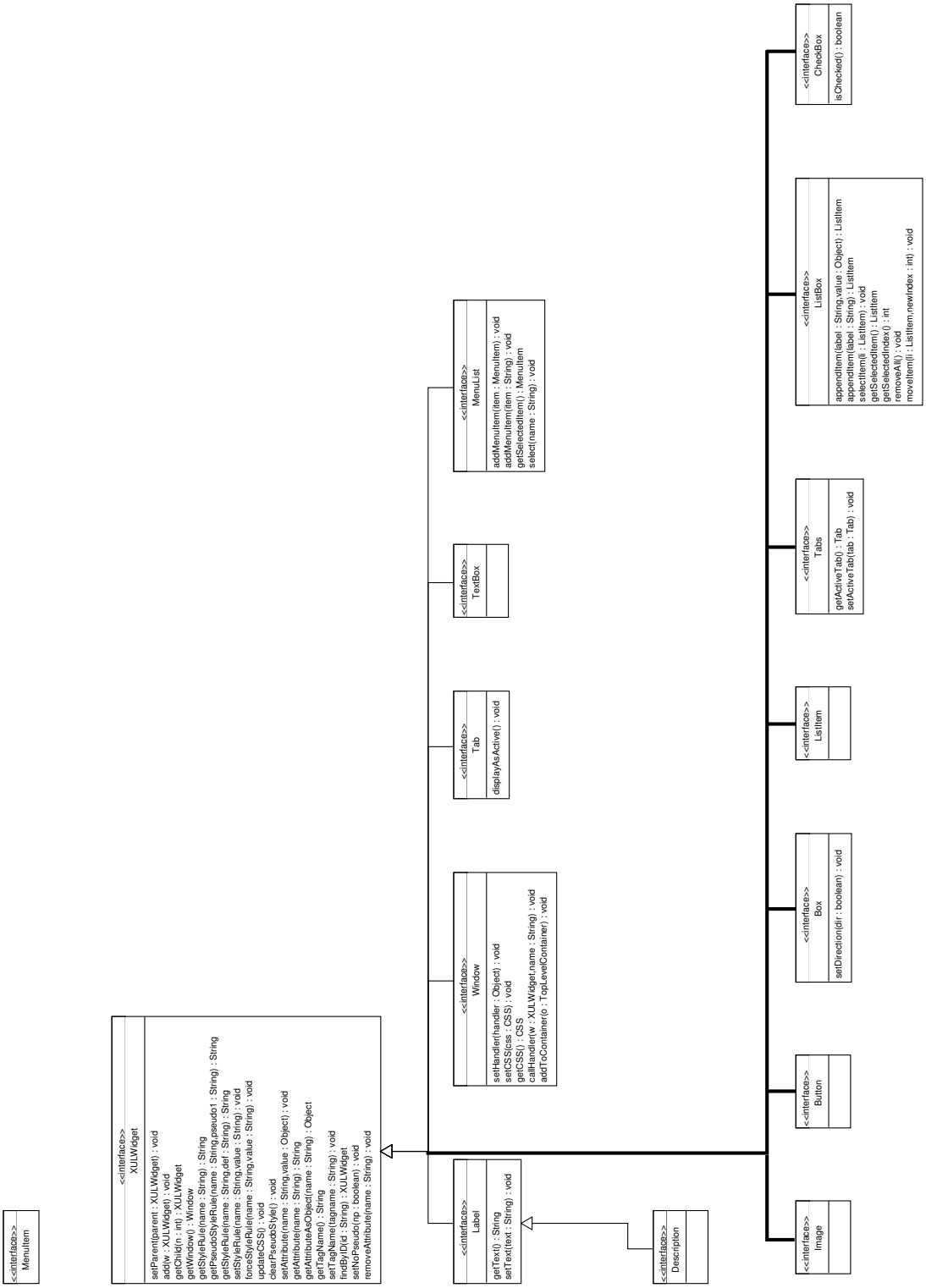

Figure 4.1.: Widgets Classdiagram

J2ME CLDC, it would require significant manually written switch blocks which increases the effort for the application developers.

4.4.2. Automatic Performance Adjustments

On slow devices or devices with a low resolution, the component may automatically disable some effects defined by CSS like *:hover*, which changes an element's appearance if the mouse pointer is within its borders. The UI component does some time measuring while updating the styles on pseudo-classes (e.g. *:hover*) and may choose to disable these if the device is not able to render them quickly enough to be unnoticed. On devices with a stylus, where the notion of a mouse and *:hoover* has no useful meaning, this effect may be disabled using a configuration option. Other costly effects and styles include rounded borders and high-quality image scaling operations. Disabling those may be achieved using build-time options for the UI component or a runtime configuration file.

4.4.3. Context / CSS changes

On changes in the CSS, which may be caused by context changes or programmatic changes on the CSS rules, all changes are propagated through all elements within a window. Every element has a fast check if a rule applies and queues all changes until the whole update is done in order to avoid unnecessary repaints or size calculations.

4.4.4. Personal Profile / AWT

The AWT implementation uses a *java.awt.Component* for each element and arranges the using standard AWT layout managers. Some layout managers are implemented which better suit the layout model of CSS. *AWTXULWidget* inherits *java.awt.Component* and is responsible for setting the inner borders of each element and painting CSS borders, interpreting text- and background colors and implementing the main parts of the previously mentioned API. Low-level events like key-press or mouseenter are also interpreted at this point. Higher level events and the rest of the paint work are done in subclasses for each element.

4.4.5. Mobile Information Device Profile

Using the existing implementation for AWT is not feasible since the UI concept of MIDP is very different. Furthermore, not all necessary classes are available in CLDC. The following problem domains were identified:

- *Classlibrary coverage* Not all classes are available and some of them are only available in a scaled down version.
- *Lifecycle* MIDP applications are part of an MIDlet which has some differences in its lifecycle.
- *Input Devices* Most devices with CLDC/MIDP do not offer a mouse, keyboard, or touchscreen.
- *Display Size* Target devices for this profile usually have a very small screen with a typical resolution of about 240x320 (see table 4.1).

Those problems and their respective design choices will be further elaborated in the following paragraphs.

Classlibrary Coverage. The MIDP does not provide a similar layout model as AWT. Therefore, this version implements a similar layout model as AWT on top of a *Canvas* using a custom layout manager. A layout manager arranges objects within the canvas and, if needed handles a scroll bar at the borders of the component. The components within the *Canvas* are then painted by the layout manager which passes an offset for the upper left corner of each component to their paint methods. All painting methods must be adapted to the new layout model of MIDP. While this would allow the same visual appearance as with the AWT version, the more complex programming and the low resources of most MIDP devices result in some limitations. Thus, many of the CSS attributes are ignored and only the most important for layout on very small screens, such as hiding elements and simple borders, are used. The further effects of the small screens are analyzed below.

Another important capability missing in CLDC is reflection. It is used to call the respective handlers for each event specified in the XUL file. Therefore, a wrapper generation is inserted into the build scripts for the CLDC version. A

wrapper is built for each XUL file and emulates some of the reflection behavior by doing a branch to a specific method which is given in a *String* argument. The wrapper generation works as described in section 5.4.6 for the virtual machine.

Lifecycle. An MIDP application has to be a subclass of *MIDlet* and, therefore, following its lifecycle contract. The application manager, which is an integral part of a device's Java implementation, must be able to control, start and stop the MIDlet. The MIDlet does also contain the necessary display access methods. One visible difference to AWT is that always only one MIDlet is active and only this one may use the screen. Each class (besides the core class libraries) needed in a MIDlet has to be packaged into a single Java archive (JAR). Without the use of optional extensions[4], it is not possible to load a file from the phone's memory.

Input Devices. High-end PDAs and Laptops do often offer a keyboard (with physical keys or a virtual on-screen keyboard) and pointer devices, but mobile phones do not contain a mouse and most likely only a numerical keypad. It has to be considered that the user interaction is different when displaying the user interfaces. Programmatically, this means that the CDC behavior through a *MouseListener* and *MouseEvents* is not available. A new event management which only requires a (joy-)stick or 4-way pad for navigation and a numerical keypad is needed. Menus will be implemented using soft-keys, available on any mobile phone. These keys change their meaning based on a description displayed above them on the screen.

Text input has to be handled, too. An XUL textfield has to be selectable and the input will then be done using the keyboard emulation of the device. Without the pointer, there is no click-metaphor. The toolkit has to emulate the click on a button by highlighting selected items (e.g. buttons) and then "clicking" them on a keypress of the "enter" key or the "select" key in 4-way devices. Highlighting an element corresponds to having the "focus" on ordinary, bigger devices. Therefore, the *:focus* pseudo class can be used to highlight an object and gives the developer full control on the look for specific devices. Using this behavior, the ordering of the elements within the XUL file is important for a meaningful focus ordering.

[4] "Optional extensions" are not available on every J2ME/MIDP Device and it is not possible to start an application using one of them if the device does not support it, i.e. there is no dynamic discovery.

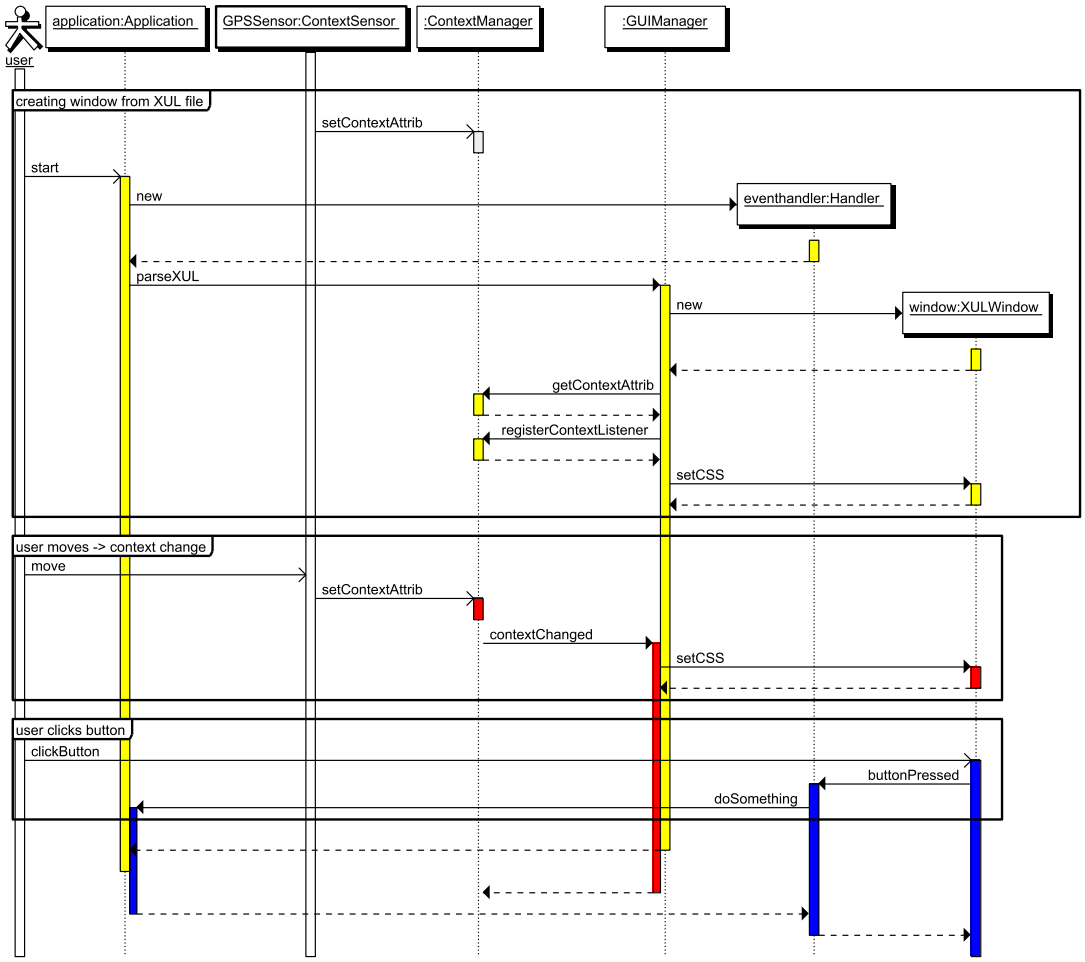

Figure 4.2.: GUI Creation Sequence-Diagram

Display Size. MIDP is designed for devices with small screens and low resolutions and, therefore, offers little space for content and graphical elements. The component offers some automatic scrolling capabilities to show all necessary information to the user.

MIDP also offers no layout manager in combination with individually styled elements. Therefore, a custom layout manager had to be built. It allows the use of the basic XUL/CSS layout model and offers scrolling if the screen size is too small.

4.4.6. Component Interaction.

Figure 4.2 shows the interaction sequence of components during the creation of a UI based on a single XUL file with context rules and multiple CSS files, the change of context and a user action.

UI Creation To create a context-aware user interface, the application developer needs to write an XUL file which points to a set of CSS files depending on the context-aware changes to the design. Then, an event handler has to be created which implements the necessary methods referenced from the XUL event handlers (see 4.3.2). The application then calls the GUI manager with the name of the XUL file and an instance of the event handler. Using some caching, the GUI manager first looks for a pre-parsed version of the XUL file which can be reused. If none is found a discovery for compiled versions of the XUL is done. If all of them fail, the XUL file is parsed and an *XULWindow* is created. All necessary context attributes are checked to decide which CSS rules should be applied to the current screen and some listeners are installed with the context manager for these attributes to allow dynamic reaction to every change. Each applicable CSS rule is installed and the *XULWindow* returned.

Context Change. In our example, the location of the user is one of the attributes the GUI manager installed a listener for. When the user changes its position, the *GPSSensor* reports a new position to the *ContextManager* which then checks its registered listeners. Since the GUI manager is registered for this event, it is called and checks if the position change is big enough to trigger a layout change. It then installs the relevant CSS file into the *XULWindow* which recalculates its layout and behavior accordingly.

User Action. When the user presses a button on which an event is installed, the *XULWindow* calls the *EventHandler* of the application.

4.4.7. Compiling XUL

String handling and thus XML parsing is an expensive task on today's mobile devices. Compilation for XUL files was therefore added to the UI component. The process of UI rendering was speeded up significantly. Compiling is done in

a similar way as parsing the XUL file at runtime. Instead of creating the objects when an element is encountered, the needed constructs to create the objects are inserted into a Java file and are then compiled using the standard *javac* compiler. Compiled classes are detected automatically at runtime to make this as transparent as possible to the developer, the framework then loads the compiled class instead of loading and parsing the XUL file. Unfortunately, loading classes from outside the application *jar* is not possible in CLDC and thus compiled XUL files are only achievable if the compiled class is available at the time of the JAR creation.

Automatic detection of compiled XULs is done using a mapping from XUL resource names to Java package name and class. Thus, the parsing method first tries to instantiate the class (*Class.forName()*) and, if it is not available, parses the XUL file. This improvement allows some easy speedup for the UI at creation time while not decreasing the code compatibility over the available platforms.

Measurements The lack of high-resolution timers on mobile devices makes exact measurements difficult. Futhermore modern JVMs are a black-box having numerous adaptive optimizations resulting in non-repeatable results. The Sharp Zaurus SL-C860 PDA was introduced in 2003 and is therefore slower than more recent mobile phones. This alleviates the lack of high-resolution timers with slower execution speed. It features a IBM J9 Virtual Machine with options to turn of dynamic optimizations leading to repeatable results. The compiled XUL files are tested on the J9 Personal Profile version of the XUL rendering backend. A UI from an XUL with 5, 10, 15, and 20 visible elements is generated and the time between the call to the UI generator and the return of the call is measured. The XUL is parsed using the XmlPullParser (Slominski 2007) in case of the non-compiled version and the CSS is parsed using our own parser in every other case. Whenever some context constraints are given for a CSS, it is only parsed if that CSS is needed at the moment and then cached.

The results are shown in 4.3. As indicated, the speed is improved by a factor of 7 with 5 elements which is a common number for mobile user interfaces and still a factor of 5 with 20 elements. The absolute time for UI creation is decreased by up to 110 msec which is noticeable even on this relatively fast PDA. With slower PDAs or mobile phones, the speed increase is even more important. In

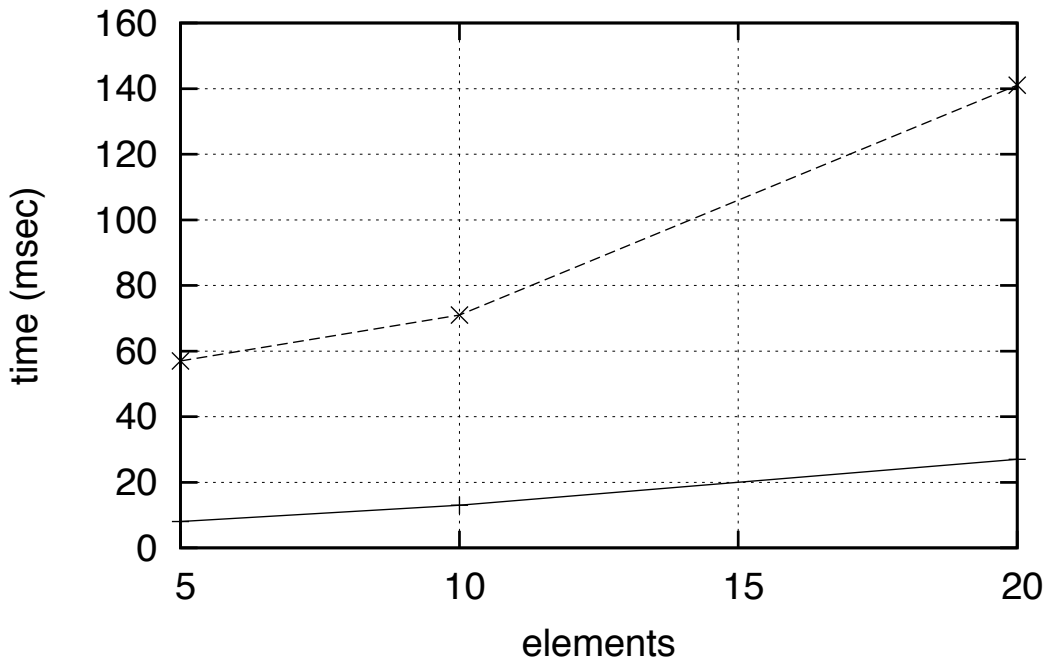

Figure 4.3.: Creation Time for a UI using the Parser and Precompiled XUL

the tested PDA and two tested S60 mobile phones the critical time of 150ms (cf. (Tolia, Andersen, and Satyanarayanan 2006)) cannot be reached without precompilation.

4.4.8. Sample Screens

In the project where this framework was developed, a prototypical context-sensitive application was created. The goal was to show that the development for different devices is much easier using this framework. Experiences during the development of the prototype showed many bugs in the rendering of different virtual machines. This would be nearly impossible to work around again in every application. Thus, the framework was helpful since the workarounds could be factored out into the UI component.

Figure 4.4 and 4.5 show the XUL mentioned above using two different stylesheets on two different devices. The changes in the CSS are minimal and the adequate CSS is chosen automatically. The latter one could also be loaded dynamically in bright sunlight in order to improve the readability.

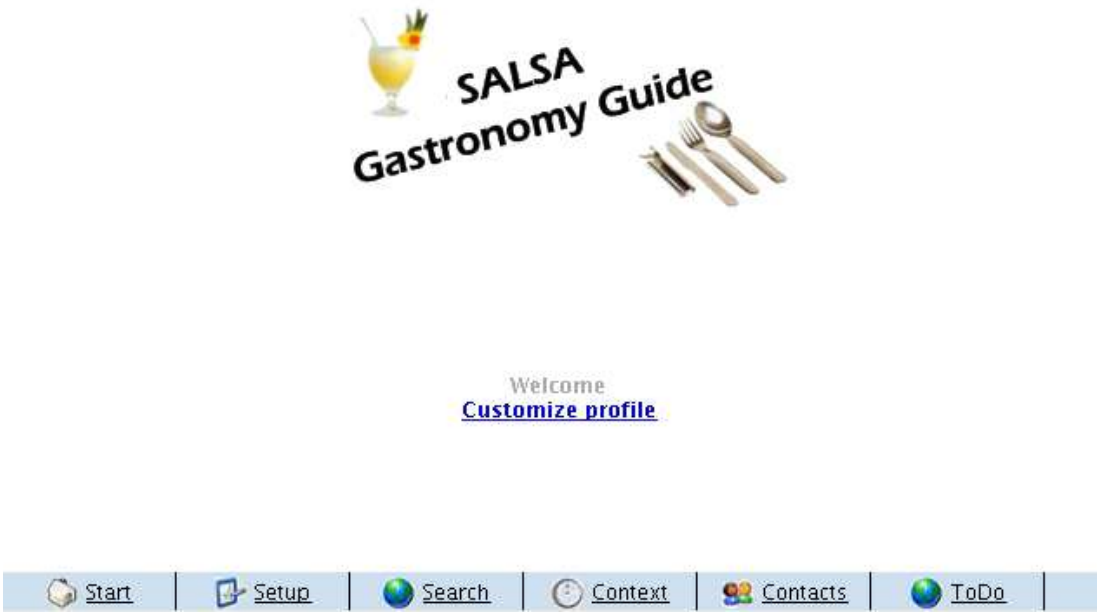

Figure 4.4.: The Application on a High-end PDA

Figure 4.5.: The Application on a Low-resolution, Low-contrast Screen

4.5. Evaluation of the User Interface Concept and Implementation

In the preceding sections, a novel approach to separate the adaption of user interfaces to different devices and user contexts, as well as the logic by using selective and automatic addition of stylesheets to the user interface was shown. The approach can easily be be used by developers and allows to style the UI for different devices by every experienced web designer without any programming knowledge. Besides that, any context available can be used to adapt the user interface of an application without giving away any information to the service.

Each of the stated requirements will be discussed in the following paragraphs.

Semi-Automatic Adaption to Heterogenous Devices. Currently, it is not possible to build user interfaces for different screen sizes or device capabilities easily. Most toolkits offer some basic widgets without any possibility to style them individually or let the developer design each UI all over for each screen size only. While HTML overcomes some of these limitations, its page based execution model does not fit the wireless use case with high latency times very well.

The XUL based approach combines the flexibility of XUL and CSS with a powerful local programming concept. It is possible to design plain interfaces that automatically adapt to every type of device while having the possibility to increase the suitability for every given screen size with custom CSS files. This reduces drastically the development effort for application with multiple target devices.

Adaptation to user context. Through context-based selection of style sheets the user interface is adaptable to every user context. The adaptation process works without any support in the components logic. It is solely based on the rules specified in the XUL file. This allows an easy creation of different appearances using context data.

Usage of private context. By splitting the appearance change by context information from the component logic, no information about the context information used in the UI adaptation will leak to the component.

Low processing needs. Benchmarking the performance of the UI component is not feasible with the high number of different Java implementations on mobile devices which all have very different performance properties. However, it can be said that XML parsing is one of the weak performance points of todays mobile phones which leads to slower UI creation than e.g. using MIDP or AWT directly. Furthermore, the elements need to be drawn using Java and MIDP may be rendered using native libraries. There are no architectural reasons why the XUL based approach should be slower than any other non-native UI component with similar styling properties. Pre-compilation of the XUL can be used to increase the UI instantiation time significantly.

Customization by the user. The CSS specification features inheritance of style elements. It is therefore possible to supply the framework with a user CSS file which modifies the default behavior of each element. Simple customization can be done using a graphical user interface which allows modification of the background picture, default font size and colors. These properties are then put into a generated CSS file which is used on every window.

5. Service Isolation and Data Protection

In context-aware services, the services or the applications attempt to better support the user by adapting the user interface or the information processing to the user's current situation. This certainly improves the usability (Kaasinen 2003) of the applications, but helping the user to find and deploy the right services or applications for the current situation is also an important point. Many services are needed to serve different usage scenarios which leads to the fact that not a single provider will be able to build all imaginable and useful services. Therefore, a mechanism to discover services and to download and execute them on mobile devices is needed. Chapter 3 introduces a concept to discover suitable services related to the user's context, while this chapter presents an approach to isolate (non-electronic) services from untrusted sources, give them access to sensible data, and still protect sensible data from leakage to service providers.

A single service provider will not be able to offer services for every imaginable situation, so an ecosystem for services is needed (Jacobson, Booch, and Rumbaugh 1999; Garud and Kumaraswamy 2002) leading to a problem of trust, especially when private information, as context data, is involved. While it is feasible to assume trust to a single service provider, most users will not trust all providers to use their data carefully and respect their right of informational self-determination. A mechanism to download components which may then use context data in their processing for user adaptation without leaking any information about private information over the network is needed. Furthermore, the integrity of each component has to be guaranteed and its interference with other components must be prevented. The later problem can be addressed using the Java Sandbox or similar concepts on other platforms. These code-based protection schemes control access to security relevant functions. Only trusted code

components are allowed to use them while any other component's access is denied. Although some protection on code basis is available in the Java sandbox, it is not easily extendable to protect data from leaving the device while still allowing calculations using that data and access to the network. For some types of adaptation, it is possible to factor out the sensitive part into some trusted components and use code-based security in the other components which prohibits access to any sensitive information. For the adaptation of user interfaces, an approach using this technique was introduced in the previous chapter. Although very secure and easy to implement, this method is only suitable in a narrow scope and it does not permit any unanticipated context-usage. Therefore, an extension to the security model which broadens the sandbox model to data is created. In our prototype, it is implemented as a byte code interpreter which runs inside a JVM. Utilizing this JVM, untrustworthy code may use all locally available data, but may not send any information originated from private information over the network.

Another problem is that in some Java configurations, which are predominant on today's mobile devices, loading code over the network is not possible. So, the virtual machine is not only necessary because of the extended sandbox, but also because the Java Connected-Limited Device Configuration (CLDC) does not permit classes to be loaded which are not located within the installed original Java Archive (JAR) file.

The remainder of the chapter is structured as follows: First, requirements for the service isolation are shown and related approaches to secure data paths are described. Afterwards, the general structure of the VM is explained. The next section discusses the data traces of the interpreter. In the last section, the implementation is evaluated against the requirements.

5.1. Requirements of Service Isolation in Mobile Applications

A platform for mobile services needs the ability to deliver any kind of (electronic) service, so the requirements in the following paragraphs are deduced from the literature or requirements of services outlined in the previous chapters.

Many, isolated services. "Users are different and they may use the services for many different tasks, even for tasks that were not anticipated in the design" (Kaasinen 2003). The number of different services implies a working ecosystem with many providers implementing components and services (Garud and Kumaraswamy 2002). Each of them has to work independently and without the possibility to disturb any other service (cf. (Czajkowski 2000)).

Low Memory Footprint. In parallel to the low memory requirement stated in chapters 3 and 4, the memory available in mobile devices is not virtually unlimited for everyday applications as in desktop computers. Therefore, the service isolation should not increase the used memory by a large magnitude.

Execution and Start-up Speed. Mobile processors are still more slow than common desktop CPUs were some years ago. Therefore, the solution may not slow down the execution of downloaded components a lot. Especially the start-up delay imposed by verification needs to be small (Bauer, Reichardt, and Schuele 2005).

Verification on the Mobile Device. The multi-provider nature of the downloadable services requires a verification on the mobile device. Further on it is assumed that the user will only trust one entity[1] and therefore this entity would need to review all components by all providers. This does not scale very well for many providers since every application needs to be reviewed and in many cases a source-code review is mandatory to find any case of information leakage. Currently, Apple is following the single provider verification approach for iPhone Store Applications and hits major scalability problems leading to a slow approval of new developers. This model also leaks intellectual property of the providers to the verification authority which limits the willingness of software vendors to cooperate with potential competitors on verification.

Access to all context data. The main goal of this work is to increase the context data usable in services under the restriction of privacy maintenance. As stated in the introduction, it is desirable to use as much information about the user context as possible while preserving the user's privacy. The usefulness

[1] The same argument would be true if it were "not many entities".

of components can be improved with access to all context data. Therefore, it should be possible to use any available data.

Privacy. The user needs the ability to specify, with any desired granularity, the access rights to context attributes (cf. section 2.3). The range of possible access rights are "no access at all", "decreased accuracy", and "full access". While it is not important to shield them from any local component, no attribute marked as private may ever get to a remote service provider through an untrustworthy component.

Network Access. Many services do need up-to-date information from a server. One example is the routing service outlined in section 3.5 which needs to download accurate map data and current traffic patterns. So, the main challenge is to allow access to all network resources while maintaining the previous two requirements: protecting the user's data while still giving full access to context data.

No false positives. A false positive, in this matter, means a wrongly detected information leakage which leads to stopped execution or a component which cannot be started in case of prior verification. Any false positive will deny access to a specific service and therefore should occur only in limited cases.

5.2. Existing Approaches

Existing approaches are from the four groups model checking, sandboxes, anonymization, and runtime verification. In each of these groups many implementations for different security policies exist. In this section only the basic approaches are outlined without going into detail for the different implementations.

Loading application logic at runtime can be done using many different approaches. In the CLDC, the loading of classes does only works with classes already present at the installation time. One way to solve this problem would be the inclusion of a scripting language interpreter like JavaScript (Flanagan 2002), Python (Lutz 2006), or TCL (Ousterhout 1994).

There are multiple methods in different areas of computer science which secure the data movement behavior of applications according to an (implicit) contract.

However, current methods are not suited for the described scenario since their computational costs are too high for mobile devices or they do not allow simultaneous usage of the network and the private data while guaranteeing the integrity of the privacy boundary.

The predominant technique to secure data in Java application is the sandbox concept (Gong, Mueller, Prafullchandra, and Schemers 1997). The sandbox grants access to resources depending on the security context of the calling methods. Using a sandbox could prevent some methods from accessing context data at all or deny access to the network. Giving untrusted components access to context data and to the network is not possible in this model without compromising the user's privacy.

Hardware virtualization uses similar approaches with a hypervisor. It intercepts system calls to the hardware and replaces them with safer wrapper functions. Those could be used to validate security rules or prohibit some functions entirely (Mitchem, Lu, and OBrien 1997).

Static Code Analysis and Model Checking (Clarke 1999) trace every possible path within a program to check the predefined contracts. These techniques tend to detect false positives since every possible path through the program is used and no knowledge about valid input can be used to narrow the analysis. Furthermore, model checking is too computationally heavyweight to be executed in real-time on today's mobile devices.

Runtime-verification is a method to check predefined test cases at runtime. Therefore, rules are specified using a temporal logic. In the context of Java, the predominant technique is instrumentation (D'Amorim and Havelund 2005) which adds some code to each class that allows tracing the method calls and their results. Adapted rules for this use case would be possible, but the runtime computing power demands are too high to be fulfilled by a mobile device and the constraint that the test cases are trusted cannot be filled.

A different approach to preserve user privacy is anonymization between the mobile device and the service provider (Tatli, Stegemann, and Lucks 2006). By using so-called "Mix"-Nets, many requests of the same user are not correlated to the IP address of the device. This can be helpful if the sensitive data which is protected by anonymization does not contain any identifiable data. If the component wants to greet the user by its name, it would be able to send the

user's name of the user along with the sensitive data. Nevertheless, this could be combined together with the method shown in this paper to further enhance the user's privacy.

5.2.0.1. Evaluation of Current Approaches

The existing approaches can be grouped into *model checking*, *sandboxes*, *anonymization*, and *runtime-verification*. Those are evaluated using the requirements outlined above. A summary is given in table 5.1.

Model Checking. Model checking performs an exhaustive search of all alternative paths through the software. This has to be done before starting the software and needs a large amount of memory and processing power to enumerate all paths. It is not suited for the verification on the user's device. Protecting access and tracing usage of sensitive data is possible, but the number of false positives is quite high since not only really used code paths are evaluated.

Sandboxes. Sandboxes and hypervisors do restrict access to methods or data based on a policy. These only restrict access to some methods and can therefore be implemented in a fast way. Services can be isolated and each does only have a low-memory overhead imposed by the hypervisor. Access to sensitive data can only be granted for the component or prevented. Usage of the data while giving network access is not possible in this model.

Anonymization. Anonymization does not change the component's behavior, but is attached at the network functions. The data sent is routed through some sort of anonymization network in order to conceal the source. Therefore, the approach is quite fast, the only computational work is the encryption of the data depending on the used algorithm. The execution of the component itself is unaltered and at full speed. The main drawback is that it is not suitable for all data and services. First of all, it requires sophisticated payment solutions to allow payment or subscription services. Furthermore, some context attributes may contain data like the name or the address that can be linked to an individual, even if the source of the transmission is unknown.

Runtime-Verification. Traditional runtime-verification needs explicit rules that are able to decide if the software adheres to a contract. These rules are based on pre- and post-conditions of code blocks. Using these rules, it is hard or impossible to find out what has been done using a specific data element, but only if something was changed or not. Moreover, most approaches do heavily use instrumentation and therefore decrease runtime performance dramatically.

Approach	Many services	Memory requirement	Speed	On-device	Data Access	Privacy Protection	Network Access	False Positives
Model Checking	0	-	-	-	+	+	+	-
Sandbox	+	+	+	+	-	+	-	+
Anonymization	+	+	0	-	0	+	-	+
Runtime-Verification	0	-	-	-	+	+	+	-

Table 5.1.: Existing Data Protection Approaches

Table 5.1 summarizes the capabilities of the existing approaches. Especially the combination of good security with high performance on mobile devices is not available in any of them. Therefore the next sections focus on a new approach combined these features.

5.3. Virtual Machine Concept

In the previous section, current approaches were outlined. These do offer a way to use private data and the network, but still make sure that the private data is secured by either requiring an offline analysis or a trusted party reviewing each component for "good" behavior. To meet all requirements shown in section 5.1, an online verification approach is needed which considers some of the advantages provided by a code analysis, while offering an instant start of the component without a trusted third party. Online in this sense means verification during execution of the component.

The VM enforces a strict version of the chinese wall security policy (Lin 2006; Anderson 2008). The chinese wall policy originates from financial auditors. An accountant may not have access to sensitive data of two competing companies. This is enforced by a virtual, moving wall which is built between the user and the sensitive data. If the user gains access to sensitive data of one company new wall is errected to the competing companies and the auditor may not work for those.

Verification techniques do traditionally check every possible path through the code of a component and, thus, are computationally very complex. To certify a component before it is executed, this is the only possible way since it is not known beforehand which path will be chosen. The concept for data protection shown in this chapter moves the verification into the virtual machine which executes the component. Combining the verification and the interpretation of code eliminates the start-up delay and limits the verification efforts to the code paths which are really used. Without the need to store data for every possible path, it also reduces the memory requirements and forces the concept closer to a feasible online verification.

Each component (or even each class or method) can be categorized into "trusted" and "untrusted". However, the aspect of classification of the classes is not within the scope of this work. Several mechanisms are conceivable, such as labeling each locally and manually installed class as trusted while every class downloaded on demand is untrusted or the Java security context could be reused (Gong 1997). However, more complex schemes could be used as those described in (Garfinkel, Pfaff, Chow, Rosenblum, and Boneh 2003), (Rubin and Geer Jr 1998) or (Zhang 1997).

Mobile devices are equipped with relatively slow processors which lead to unacceptable slow applications in an interpreted VM enviroment. Therefore, our approach executes the trusted code in the device JVM directly which is significantly faster and only the untrusted code is interpreted. The interpretation is fully transparent to the application and trusted code can access untrusted code and vice versa. When common code, like a UI framework or high-level context processing (cf. chapter 3) functions are included in the trusted code, the slowdown of the whole application can be neglected. In this chapter, the VM that is

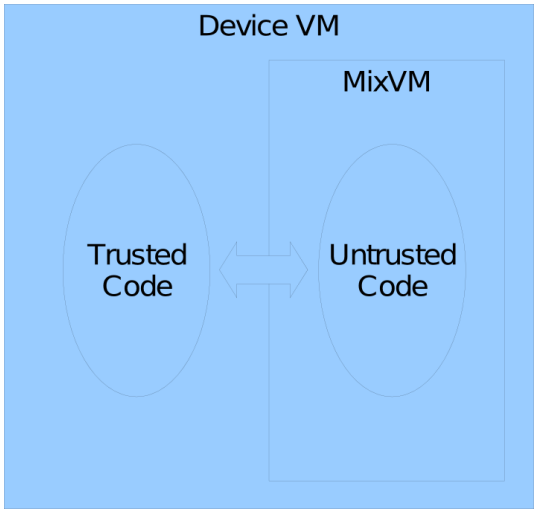

Figure 5.1.: VM

shipped with the mobile device is refered as “device VM” and the VM running inside as “inner VM” or“MixVM” (cf. figure 5.1).

Each variable (object or primitive type) within the inner VM is tagged with its *status* which indicates to which group of data the variable belongs. The group determines if the content of the variable is sensitive and if information about its content may leave the mobile device over the network. At the execution of a Java bytecode instruction, the tags of the involved variables are checked. Depending on the status of the variables which influences the results of the instructions, the tags for the result are set. As some instructions do also influence the program counter (PC), it must be tagged in the same way as if it is influenced by a tagged variable.

Initially, variables get their *sensitive* status when they transition from trusted code to untrusted code in a wrapper. The status is then checked again when trusted code is called which accesses the network in order to form a boundary to the network.

5.4. Design Considerations for VM

The concept for data protection shown in the previous section can be applied to an interpreted JVM as well as in a just-in-time compilation JVM. A hybrid approach was chosen to minimize the development effort and to ease the evaluation. It is

an interpreter within an existing JVM for the untrusted code segments. While the interpreter written in Java slows down the execution of untrusted code, it has the advantage of running within existing mobile phones and PDA.

Furthermore, this adds the ability to execute downloaded code on CLDC environments as a side effect. In Java, a class loader is responsible to load classes from a file or over the network into the virtual machine. The class is only available within the context of its class loader and all its parent class loaders. Within CLDC, a simpler security mechanism is implemented which prohibits custom class loaders and does only contain one which limits its access to a single JAR file. The interpreter presented in this prototype may be used to overcome this limitation on current devices.

This VM aims to be compatible with the Java Virtual Machine Specification (Lindholm and Yellin 1999)[2] and therefore to be compatible with a native JVM for all applications. To save the effort of implementing the class libraries again and to speed up their execution, the interaction of classes running in the native VM and the interpreted VM are fully transparent.

To load an interpreted class from a native class[3], it would use the newly implemented Clazz.forName() in contrast to the conventional Class.forName() to instruct the interpreter to load the class and return a reference to it. Every class referenced from within an interpreted class would have a special class loader which first would try to locate the class within the interpreter and then would fall back to the native classes and the bundled class libraries. This fosters a faster execution by having a framework with high level function within the native JIT or interpreter and just using the MixVM interpreter for a smaller piece of the code. In this case, a framework for context-aware mobile applications which implements the context sensing as well as some reconfigurable graphical user interface (GUI) based on context changes and downloadable code offering some service downloaded on demand, depending on the user's situation. The downloaded code does only contains some logic which generally is not as computation intense as the GUI functions or the regular context sensing. Furthermore, the untrusted source of the component does need special handling.

[2]The only incompatibility with the specification is the deferred validation of loaded classes, but this does not affect valid classes.

[3]For a differentiation of "native" and "interpreted" class see figure 5.1

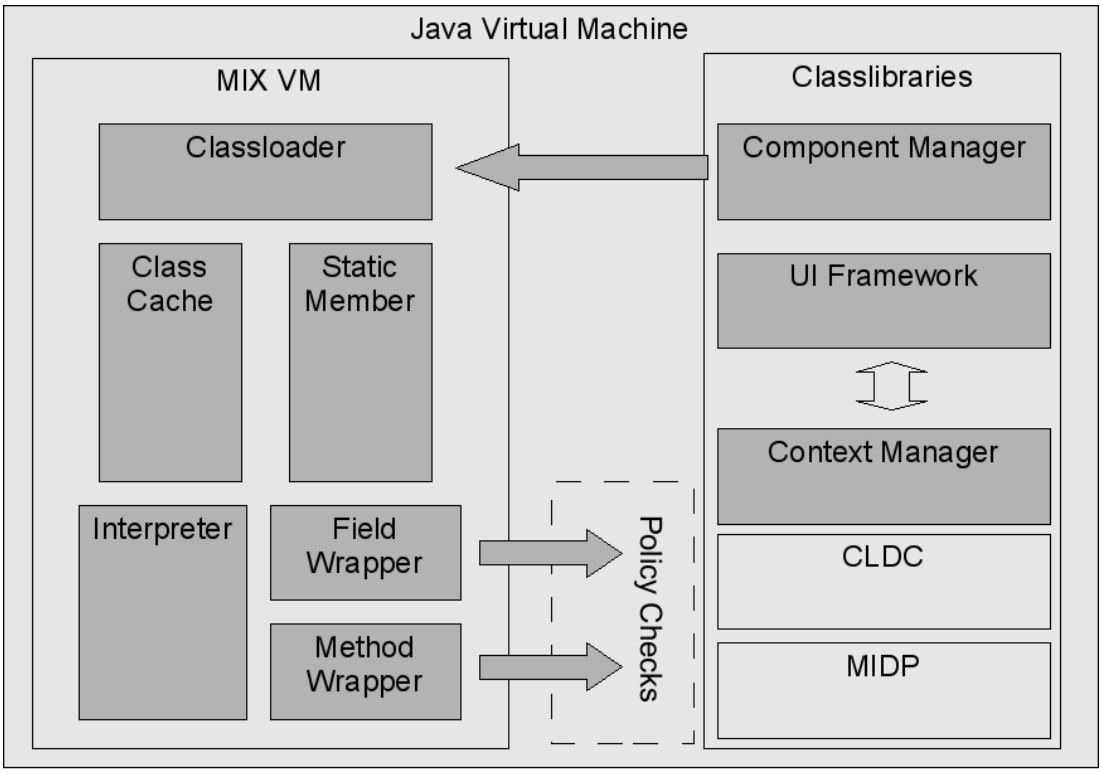

Figure 5.2.: Interaction between MixVM and the Class libraries

Figure 5.2 shows the interaction of the virtual machine with the class libraries and the native-VM. While previously described components of the framework, as described in chapters 3 and 4 as well as with the class libraries of the used Java profile, are executed in the native VM, the downloaded code resides within the MixVM. This consists of a class loader, the interpreter loop, and storage for the class and instance variables. The objects can be accessed from outside via a reference to a proxy object. Access from within the MixVM to the outer VM is performed using a field and a method wrapper. Depending on the state of the involved data and the called method, the access may be denied and the state of return values is set. Details of this process are shown in the next sections.

In the following sections, a prototypical implementation is presented. Selected design choices of the implementation are discussed in the next sections.

5.4.1. Class Loading

Whenever a class has to be loaded, the class loading process starts by fetching the class out of a JAR or downloading it over the network. A preset class path is used for searching similar to the process in a standard JVM. The class file is then parsed according to the Java Virtual Machine Specification(Lindholm and Yellin 1999). The linking and verification step, as required by the standard, is postponed to the actual invocation of the classes, methods, or constructors to save execution time. The outer VM performs equivalent runtime checks anyway. After parsing and immediate invocation of all static initializers of each loaded class, they are stored in a hash table to be found later. Various hash tables may be used if some kind of separation between the classes of different components is necessary or multiple class loaders must be emulated.

5.4.2. Interpreter

Whenever a method of a class that has been loaded by the interpreter is invoked, the method *execute(String methodname, String descriptor, OperandVector args)* is called including the method name, the descriptor, and all parameters in the Clazz instance of the loaded class. A new stack frame is created which contains the program counter (PCount) and the number of slots for local variables as specified by the method. The entries $1 \ldots n$ of the local variables are then initialized

with the n arguments of the method according to the specification. Furthermore, a *status* for the PCount is stored for data tracing (see 5.4.7). This information can be used together with the tagging of variables to detect if an instruction may leak sensitive data.

The interpreter itself is an interpreter loop which decodes each bytecode and completes the necessary actions while making notes about the status of the used data as shown in the next sections.

5.4.3. Primitive Types, Objects, and Arrays

Storage of primitive types and objects is done in wrapper classes. Each variable is wrapped into an object which contains the primitive type or an object reference. Furthermore, it contains a status indicator to signal if the variable contains sensitive information. For every primitive type, a wrapper class exists. Using these wrappers, type checking is done explicitly by the native VM. Arrays are wrapped too and every array element is also wrapped to maintain the status on an element level.

5.4.4. Instances

Instances of classes loaded with the MixVM are of the type *Instance*. They enclose a Clazz field which is a reference to the loaded class and an array containing the instance variables. The latter can be accessed from the instance with wrapper methods.

5.4.5. Threads / Locks

The behavior of threads and locks is defined in the Java Language Specification. The Virtual Machine runs within a Java environment, so it is possible to reuse the thread implementation of the native VM. Whenever a thread is created within the interpreted code, it is given to the outer VM and a new native thread is created. Methods are then interpreted within the newly created native thread. The same is true for locks. If interpreted code acquires a monitor the VM acquires a lock on the wrapped object in the native VM.

5.4.6. Accessing Outside Fields / Methods

Classes running within the inner VM may call methods of classes running in the outer VM. Java identifies a field by the name of the class, the name of the field, and a descriptor for access from the bytecode. The descriptor encodes the type of the field. A non-static field is identified by the same identifier and, additionally, a reference to an object. In an environment that supports *java.lang.reflection*, it is easy to get and set the values of these fields using reflection. Since no reflection is available in the CLDC environment, a wrapper is needed.

In the presented approach, the wrapper has four static methods: *put, get, putstatic, getstatic*. The *put* methods use the fieldname, the classname, the descriptor, and a value to put into that field. In case of the non-static method, an additional object reference of type classname or a subclass is used as argument. The *get* method uses the same identifying attributes and returns the value of the field.

Similar to the fields in Java classes, methods are identified by the method name, a descriptor, the class, and, in the case of a non-static method, a reference to an object. Access to private or protected classes in the outer VM is not possible because of Java security restrictions. The class in the inner VM will not be in the same package as the class libraries. This should cause no problem since well-behaving software components should never use the same packages as the framework. The wrapper for methods gets the identifying attributes as *String* and a *List* with the arguments wrapped into the internal representations. The method wrapper then looks for the corresponding method and casts the object to the required type. Thereafter, the arguments are unwrapped and casted according to the method descriptor. Finally, the method is called and the return value is wrapped again in the appropriate operand container.

Due to the fact that hundreds of fields and methods of the class libraries need to be accessed, it is not feasible to manually edit this wrapper. Therefore, a code generator consisting of two parts was implemented. The first one scans the class library and the trusted classes and writes the results to an XML file. A UI application parses the method definitions and offers the users a simple interface to clean the file from methods which should not be accessed via a wrapper. Furthermore, methods can be marked to indicate that they return sensitive data or

leak data to other objects. A second tool then generates two Java files. The first one for the fields and the other one for the methods implementing the wrappers.

There are several possibilities to divert the appropiate method. The following four alternatives were evaluated:

- One large method with a string comparison for each class/method/arguments combination.
- A nested string comparison, only comparing the method name in the top-level and thus reducing the number of *if* instruction in the top-level. In the next level the method name / signature is checked. The number of string comparisions is therefore decreased by a factor of methods per class on average.
- One large switch block over the *int* hash of the concatenated class/method-/descriptor.
- A nested switch: the top-level switch uses a *hashmod*256 over the same hash and then another switch for the whole hash inside the case block. The reason for the last method is that dense case operands are implemented using a *tableswitch* opcode which is much faster than a *lookupswitch*. *Tableswitch* is internally a table having two columns. In the first column, the value to compare is stored and in the second column the address which is used as the new program counter is filed. The VM has to compare each value until the right one is found. The *lookupswitch* instruction is only available for consecutive values. Here, a starting value is subtracted from the value and the result is used as a pointer to the right offset. No value comparision is needed in that case, but in case of large holes between the values the needed memory for the table becomes too big and there is an architectural limit in Java for a bit less than 65000 values (depending on the size of the following instructions).

To get some empirical data for this decision, three different implementations were tested [4]. The first one is the nested string comparison, then the switch statement over all hashes, and as implementation number three the *hashmod*256

[4]It was already shown that the plain comparison is slower if the average number of methods per class is bigger than 1.

switch with nested *if* statements for the *hash*. The test was performed on a 1.5Ghz Pentium M with 2GB of RAM and a Java 1.6.0-b09 on GNU/Linux. The VM was used in interpreted mode, since JIT mode is not predictable. In the HotSpot JIT, the decision for compilation or interpretation is dependent on many unpredictable outside factors. The static field wrapper with 1000 fields was used and called 100.000 times with random fields. In Test A, only the upper half of fields was used, Test B the bottom half and Test C had a uniform distribution over all fields. The results in table 5.2 show the average time over 5 runs. Since the nested hash approach provides by far the best performance, it has been used for the wrapper implementation.

Implementation	Test A	Test B	Test C
String Comparison	30s	36s	34
Hash	13s	14s	14s
Nested hash	8s	8s	8s

Table 5.2.: Look-up Benchmarking

5.4.7. Tracing Data

Each wrapper object has a status field indicating if an object contains sensitive data or may be passed around freely. Furthermore, a code status flag exists which indicates if the current code path may give away information about some data. Each instruction in the Java virtual machine consists of an opcode, determining the action and some optional data. Depending on the opcode, this status field is modified. The opcode classes are described in the following paragraphs. The categorization from the JLS §3.11 is used. To denote the status of a field or variable A, the notation $s(A)$ is used.

5.4.7.1. Load and Store Instructions

Load and store instructions are used to move a variable from the operand stack to a local variable or vice versa and to move a constant to the operand stack. In each of these storages, the variables are wrapped inside a wrapper object which stores the status of the variable. The reference of the wrapper is simply copied. If the PCount status indicates a sensitive block, the status is combined with the code status using a binary logical disjunction.

5.4.7.2. Arithmetic Instructions

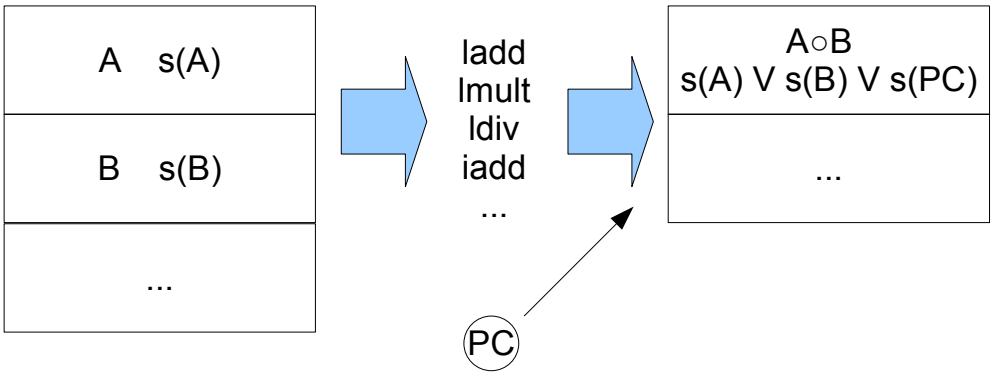

Figure 5.3.: Arithmetic Instructions

Arithmetic instructions operate on primitive types. They take one (e.g. A) or two (e.g. A,B) variables of the same type off the operand stack and return the result which is a simple arithmetic operation on the stack. The privacy status of the result (e.g. C) is computed using a bitwise disjunction operation of both status and the code status or the PCount ($s(C) = s(A) \vee s(B)$). The result is then put into a new wrapper object and put on top of the stack.

5.4.7.3. Type Conversion Instructions

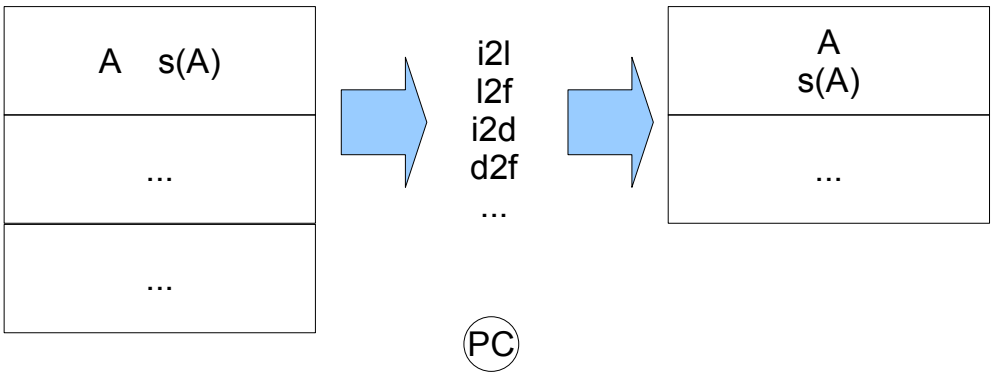

Figure 5.4.: Type Conversion Instructions

Type conversion instructions operate on a single primitive type on the operand stack and put back another primitive type. In this case, the privacy status is copied from the argument to the result.

5.4.7.4. Object Creation and Manipulation

Newly created objects and arrays inherit the status of the code by which they are created.

Array load and store operations behave in the same way as the load and store instructions. *instanceof, checkcast, and arraylength* get the status of the array/object and the codestatus combined using a bitwise disjunction.

5.4.7.5. Operand Stack Management Instructions

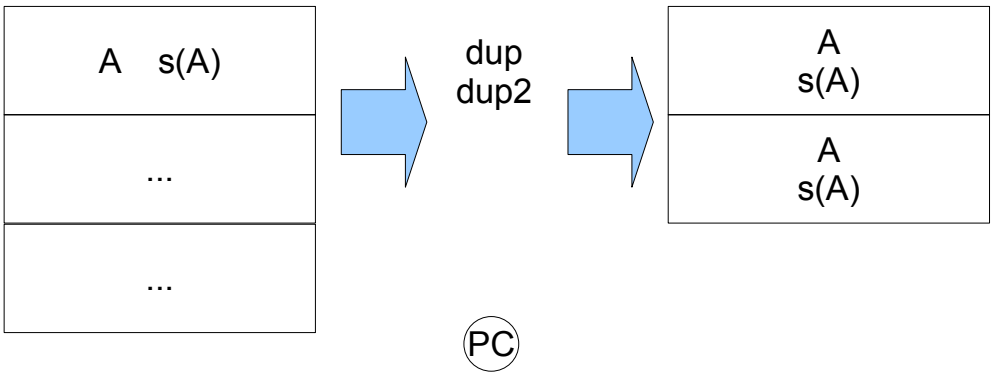

Figure 5.5.: Operand Stack Management Instructions

In case of the *dup* instructions, the newly created object gets the ORed codestatus and status of the source variable. The *swap* instruction gives BOTH variables the combination of the codestatus and status variables using bitwise disjunction.

5.4.7.6. Control Transfer Instructions

If the comparision of a control transfer instruction does not contain any private tagged variable, no special action is required. Otherwise, an analysis of the not-followed code path is necessary:

- If the path does only contain primitive types and arithmetic operations, these variables are tagged.
- If the code contains other instructions, a static analysis is started of the chunk in case of small chunks or the PCount status flag is set.

Without this analysis, the following code snippet could leak information of the sensitive variable a to b even if the comparison is not *true*. The variable b is equal to 1 if a is not equal to 2. In this case, b would implicitly include some information about a even though the assignment does not happen within a sensitive block.

```
b = 1;
if(a == 2)
   b = 2;
```

These are the main scenarios in which the VM could lead to false positives and developers should therefore be advised to use constructs like these in a cautious way.

5.4.7.7. Method Invocation and Return Instructions

Opcodes out of the *invoke* family first check if the defining class of the method is loaded by the interpreter. In that case, the *execute(...)* method of the loading *Clazz* instance is called and the *codestatus* is inherited by the invoked method. Otherwise, the corresponding method wrapper is called. Here, the suitability of the parameter's privacy status is checked and the method called.

The *return* instructions remove the current stack frame and return to the next higher frame.

5.4.7.8. Throwing Exceptions

Throwing an exception has two differenting consequences depending on an enclosing *catch* or the methods *throws* declaration.

- The exception is caught within the current method. Then, the behavior is similar to that of a Control Transfer Instruction. If the *throw* depends on a sensitive variable, the PCount status flag is set and the handling matches.
- The exception is caught by a method higher in the stack. Here, a permanent PCount status is set.

5.4.7.9. Synchronization

Synchronization does not affect any data, thus no special handling is needed. The locking mechanism of the native VM is used in that case.

5.5. Evaluation of the Concept and Implementation

The outlined concept and the prototypical implementation shown in the previous sections are the first which fulfill the requirements in section 5.1. Each of the requirements is shown in the following paragraphs.

Many, Isolated Services. The Java class file interpreter facilitates a natural isolation of the components. Each interpreter instance has its own set of loaded classes and static variables, so each component has its own separated execution environment.

Low Memory Footprint. The interpreter's size for the compiled class file is about 100kb, which poses no real problem for today's mobile handheld devices. It would be a problem for minimal sensor nodes, but those do not need this kind of downloadable services anyway. Beyond that, the heap size increases depending on the type of used primitives and objects. In our tests, it was a factor of about 1.5 to 2.5 for each object hold by the inner class, but these numbers depend on the type of VM used and cannot be measured reliably.

Execution and Start-up Speed. Using the MixVM, no start-up delay for verification is needed as in any other approach which fulfills the privacy requirement. Distribution of the trusted and untrusted code between the two virtual machines allows trusted code to be executed much faster than the untrusted code and, therefore, offer a higher execution speed together with the user interface framework of chapter 4. Given that the execution of untrusted and trusted code is handled differently, existing benchmarks are not significant. A simple untrusted class which calculates the first 10,000 primes several times is slower by a factor of 17 on an x86 computer with Sun JDK 6 and therefore would be too slow for real world usage. Benchmarks using the B2C applications in chapter 6, where the UI and context framework was trusted and the downloadable components

are marked as untrusted, showed only a slowdown with a factor of 1.5 to 3.3 depending on the usage. Using a JIT or integrating directly into a JVM instead of the interpreted Java prototype, nearly the same speed as without that concept is attainable.

Verification on the Mobile Device. The number of services prohibits a central authority verifying all component. Therefore the verification step has to happen on the mobile device. In this approach all code is verified at run-time and no assumptions about the downloaded code are made. Therefore, no trust in a central authority is needed.

Privacy & Access to all Context Data. Downloadable, untrusted components may use all sensitive context attribute, even for adaption to the user's situation. The traceability of private context data allows the access to all data until it is tried to send it over the network.

Access to network. The tracking of sensitive data fosters a separation of the context usage and the network usage. Thus, it is possible to give network access to the components while still preserving the user's privacy.

False positives. False positives are possible using this approach. Similar techniques as in static code analysis are used to find the path of the used data but not all code paths are really executed. In a static code analysis all possible values for any variable and the resulting code paths are evaluated. In this VM only the used paths are evaluated and only with the live data. Therefore, less code and data is flagged as confidential. So, the number of false positives is as high as in static analysis in the worst case, but much better in general.

The shown approach is superior to all other approaches since none of them is able to fulfill all stated requirements. One drawback is the potential for false positives, but it is still lower than in in the best competitor (static code analysis) which would not allow a decentralized service provider scenario.

6. Examples Validating the Utility of the Framework

In this chapter, two sample applications implemented using the framework, developed in the previous three chapters, are presented to show the feasibility of context-aware application development using this framework and its rapid prototyping capabilities. The first one is a consumer application comprised of several services emerged during the SALSA project. The second one a simulated business application for the construction industry.

6.1. Consumer Application

Any usage of a mobile service consists of two parts. First of all a suitable service has to be found and then the service is used. In some cases the service is already known and simply has to be activated, but having the ability to find a service in a new situation is important. Besides entertainment and navigation the largest portion of services consists of information services (cf. chapter 2). Context in informational services can be used to augment explicit user input. An example of heavy context use can be found in recommendation services which use a large set of context data to deliver good recommendations. This section shows a bargain hunter and a restaurant recommendation service as examples for this service type. Results can be easily transferred to other recommendation services. The utility of the framework is shown using several different services highlighting the reusability.

In contrast to typical desktop applications, mobile applications show different characteristics arising from the influence of their environment on their design. The key distinguishing characteristics include mobility, resource limitations, heterogeneity, personalization, and different requirements of usability patterns. In

view of all these constraints, it is even more important to provide mobile users with enhanced support for finding suitable services in an efficient way. Long-winded interactions between humans and mobile devices in the process of service discovery should be avoided since users are restricted in their input capabilities and are mostly not willing to enter large search requests, as they would do using browser-based search engines on their desktop computers. Context-aware service discovery delivers significant added value since it considerably reduces the required interaction and delivers personalized, precise search results that are suitable in the user's current context.

In the SALSA project (Software Architectures for Location-Specific Transactions in Mobile Commerce)[1], the Mobile Business Research Group at the University of Mannheim has built up a generic software platform which supports the development of context-sensitive discovery services for service brokers and providers. Furthermore, SALSA offers a client framework supporting the development of generic client applications for mobile devices that enable the dynamic integration and execution of discovered services and especially considers the user's privacy for contextual information.

The basic scenario adopted in the SALSA prototype assumes a mobile user equipped with a mobile device. The mobile user is interested in immediately getting value from physical (business) services suitable to his current context. Thus, he sends a service request to a centralized service broker which delivers a choice of electronic services that he can use for further exploration. The service broker preselects services that fit to the user's current context (e.g. location, time, weather, user profile). The mobile user selects one of the specialized services by category and, if necessary, the client application automatically downloads and executes components that are needed to consume the service. When sending a service request to the specialized service using the implicit available context information of the mobile device, the user is returned a personalized, and customized list of services suitable for the submitted context. After choosing a service provider, the client may download components for the service. The general working principle of the scenario is shown in figure 6.1.

[1] http://www.salsa.uni-mannheim.de/

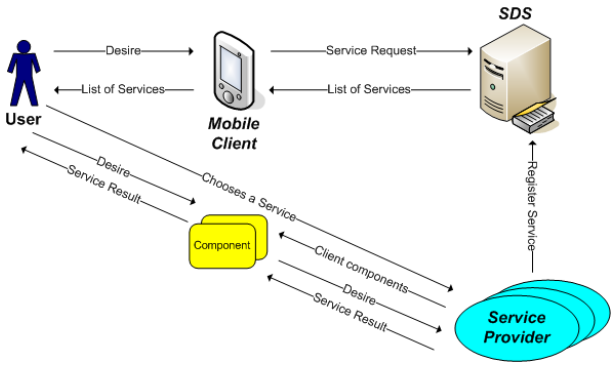

Figure 6.1.: Scenario

6.1.1. Service Discovery

The application does consist of the aforementioned framework with several context sensors connected to the context manager. The following sensors were implemented and used in this example:

- *Calendar* to report the available time slot and the next location of the user,
- *GPS and WLAN positioning* to accurately report the position of the user,
- an interface to a *weather service* reporting current weather at the position of the user,
- *device orientation* to adapt the user interface, and
- *context sharing* with instant messenger context to gather position information of friends and colleagures.

The user handles the application through a main screen (see figure 6.2 for an example rendering on a high-end PDA) which allows to search for other services and set user preferences, including the context access permissions.

This screen offers the following actions to the user:

- Setup: Setting profile data and personal preferences.
- Search: Discovery of suitable service for the current user context.
- Context: Overview of the currently gathered context attributes and their visibility for services.

Each function will be described in the following paragraphs.

Figure 6.2.: Start Screen with Menu

Settings. The settings screen gathers configuration options from installed services. To improve personalization, the user has the possibility to specify some user attributes, as the preferences towards special food or whether he prefers credit cards or cash. The dynamic preferences screen can be seen in figure 6.3. More options can be added dynamically via downloadable components linking into this dialog.

Context and Privacy Settings. On the screen in Figure 6.4, the context attributes available to the application are shown together with their privacy status. The user has the possibility to mark each attribute as “public”, “blurred”, or “private”. It is then set in the context manager so that the access of the other components’ access will be denied or degraded. “Private” data may only be used locally to improve search results or the graphical user interface.

In the shown example, the user’s position is blurred and therefore its accuracy is degraded artificically.

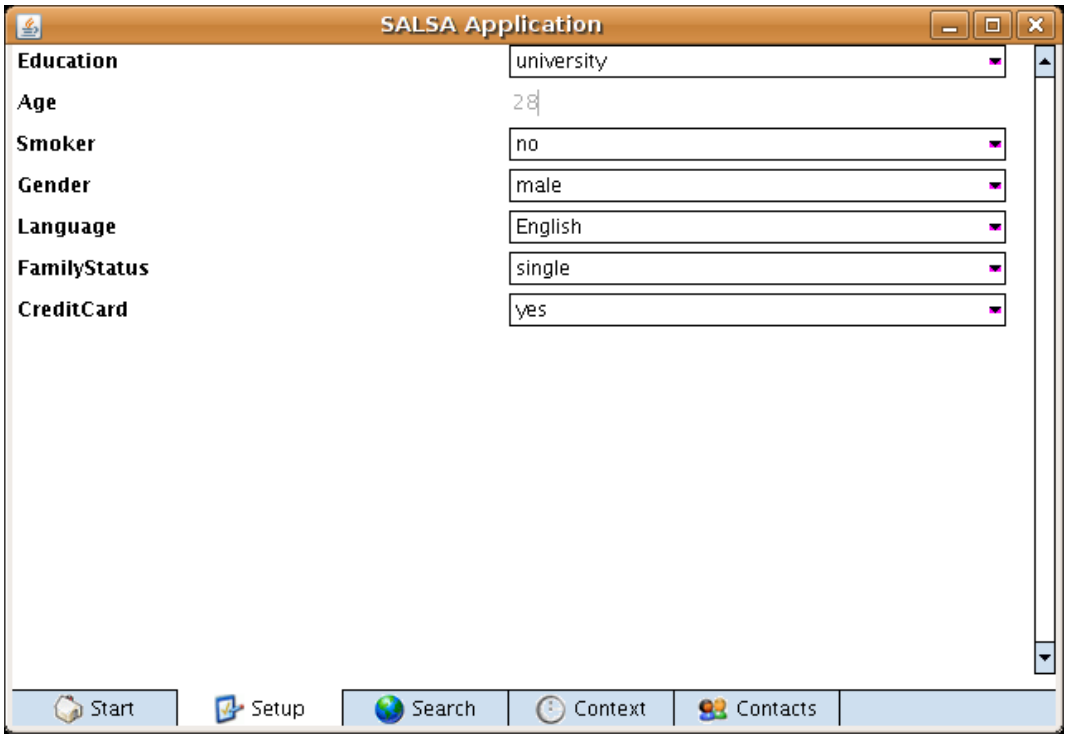

Figure 6.3.: User Preferences

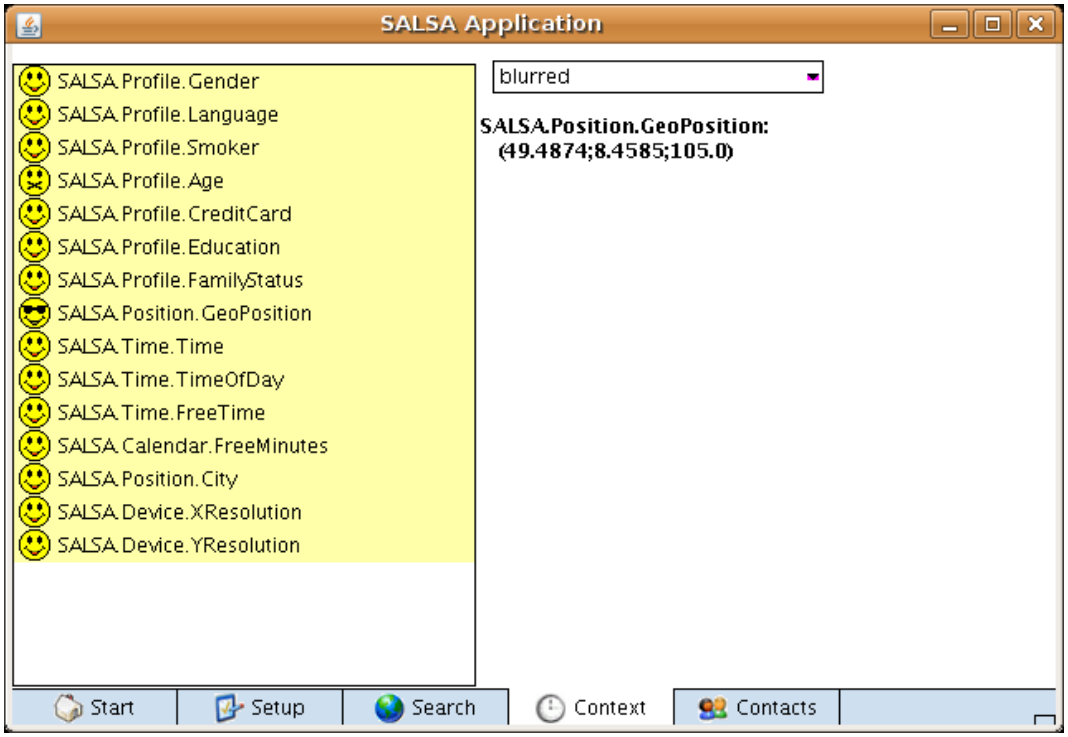

Figure 6.4.: Context Attributes Visibility

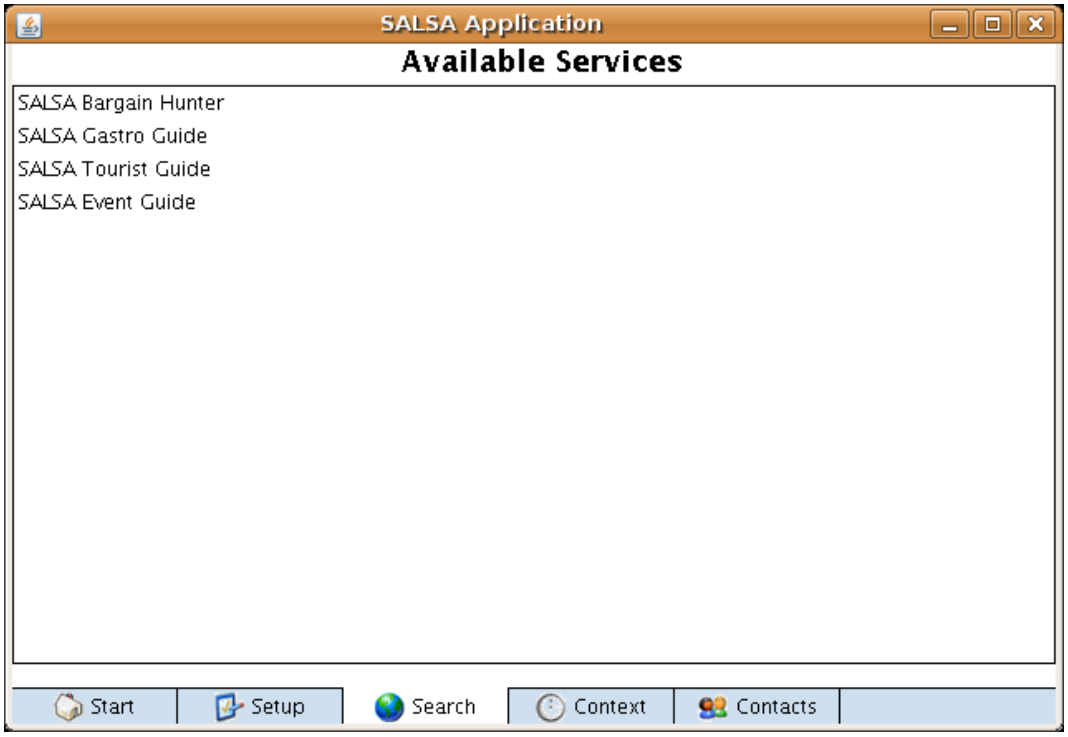

Figure 6.5.: Service Selection

Service Discovery. The invocation of the service discovery function in the main menu sends the implicitly gathered public context attributes together with the explicitly entered attributes in the serialized format to the discovery server of chapter 3. All services in the database are compared to the set of context attributes and those which contradict the attributes are eliminated. The remaining services are sent back to the client and the private attributes are used to refine the search. If the number of services (which should be) sent over the network is too large, a warning is sent to the user instead. The user also receives a notice on which additional context attributes can limit the number to a reasonable amount. Using this information the user may decide to resubmit the request giving out additional data or using the truncated result set.

In the next dialog, the user is offered a list of services suitable for his current situation. A choice to get additional information about each service or to use one of them is given. Upon selection, potentially needed components are downloaded, instantiated, and executed within the isolating virtual machine pictured in chapter 5.

For the sample application developed during this thesis, four services were implemented. Each of them used the service discovery, the UI toolkit, and was downloaded executed within the MixVM.

6.1.1.1. Gastronomy Guide

A gastronomy recommendation guide is a typical information service which is highly dependent on the current context of the user. First of all the choice of restaurant depends on the location, but also on available time for lunch or previous choices. Thus a guide was developed enabling its users to find restaurants and bars in the vicinity. Following the selection of the service, the application downloads a component which contains the user interface elements and further required components like the routing component and a security module if not already available on the device. Furthermore a restaurant may offer additional services such as table reservation or special prices.

On the initial screen of the guide, the user may choose additional criteria for his restaurant search, like the maximum distance or the preferred cuisine (figure 6.6). Afterwards, those are sent with the context attributes to the service provider. The server uses the same search concept as in the service search above, but implements a different mapping between the restaurant description and the applicable context attributes. On the device, the same type of filtering is performed. Mappings include, amongst others, those from the address to coordinates, indoor/outdoor facilities, and opening times to time restrictions.

Restaurant Details A click on a list item in the results page leads to the details of the restaurant or bar. Anything included in the restaurant descriptions, besides the context rules, is displayed. Furthermore, a button to calculate a route to the place from the current position is offered.

Routing to Place of Interest. Depending on the privacy settings of the user's position, two types of routing are possible. In the case of private coordinates, the routing is performed as described in section 3.5. Otherwise, the service providers do provide a map from the user's current position to the destination in order to save processing time and bandwidth (figure 6.8).

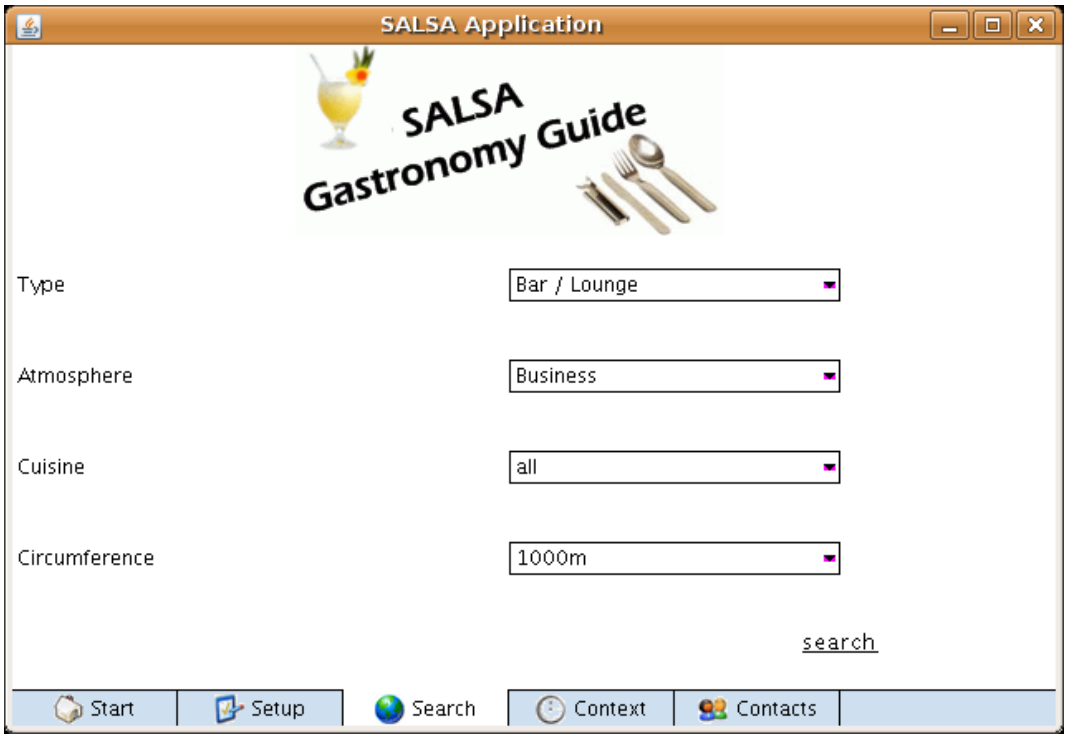

Figure 6.6.: Gastronomy Guide Settings

Figure 6.7.: Gastro Details

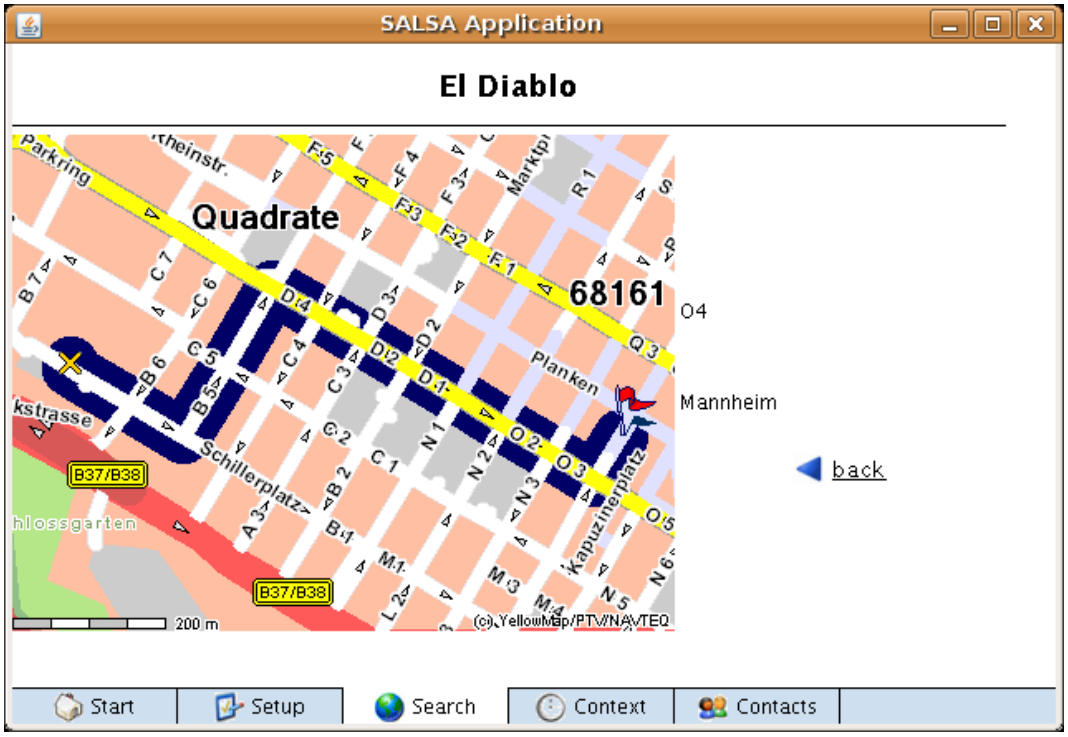

Figure 6.8.: Navigation to Restaurant

6.1.1.2. Bargain Hunter

The service "bargain hunter" enables the user to find bargains matching his personal preferences in his vicinity. The local component on the device allows the search for coupons and storing them in order to show them to the collector later. The coupons' personalization or adding barcodes for scanning is also possible.

6.1.1.3. Event Guide

Using the event guide, a mobile user may find relevant and interesting events in his current neighborhood available to his profile. Amongst the explicit preferences, other attributes like the current time, the position, and the available free time in the calendar are used to find suitable events. On the server, the current weather is added to further refine the results.

6.1.1.4. Tourist Guide

Additionally, a tourist guide was implemented. Resembling the gastro guide, places of interest in the city the user currently resides are shown. Besides the routing to one destination, the tourist guide is able to calculate routes including all places.

6.1.2. Conclusions

The services shown in this chapter proves the feasibility of implementing typical context-aware systems using the presented framework. Other projects[2] always limit either functionality or privacy protection for these types of services. All services used only the inference capabilities of the service discovery (cf. chapter 3) for any context based decisions.

All components shown in the chapters 3, 4, and 5 were used and the effort to develop the services was mainly in the design of the user interface using XUL and the transformations of the service descriptions. Only a very small amount of actual Java code had to be implemented for each service (in the range of 100-200 lines of code per service).

Each of the services was tested on multiple devices with different input and output capabilities and could be used without any modifications which shows the capabilities of the UI framework.

6.2. Mobile Support Application in the Construction Industry

The usage of new mobile applications for mobile workers is still quite low. While technological problems decreased, in recent years, the overall benefit of using mobile technologies is not clear in most cases (Gebauer, Shaw, and Zhao 2002). In the project, GEM, methodologies to measure the possible benefit of different mobile solutions and technologies were searched. Therefore, different variations of technologies and improved processes had to be tested. The construction industry was chosen as an example since it is inherently mobile and very labor-intensive.

[2] cf. the respective sections in the chapters 2, 3, and 4.

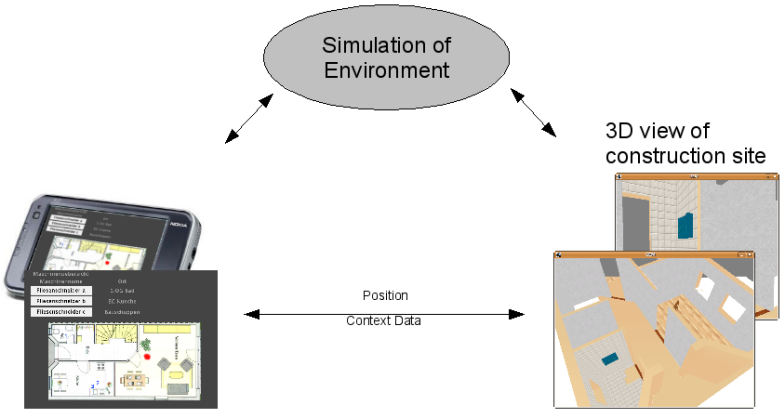

Figure 6.9.: Laboratory Setting

A research model was constructed including the constructs of information gathering and information access costs (Deibert, Heinzl, and Rothlauf 2008). This model needs to be tested using a combined experimental and simulational approach.

To validate this model, it is necessary to be able to easily test many variations of the same process. It is hard to find a controlled environment offering the possibility to change the setting often. Especially the integration of the mobile application in the backend information systems is not possible for many different variation. Therefore, a system is needed which allows easy creation of mobile applications. Furthermore, different levels of sensoring support should be evaluated. It is nearly impossible to build and access those in a real construction site environment because of technical and legal hurdles. Additionally, it is hard to test different process variations on-site in a real environment, so a laboratory to gather empirical data which can later be used in a simulation model to predict the benefits of different mobile technologies had to be built. The laboratory

combines a virtual 3D view of the construction site with a mobile application. Context data is transferred from the coupled virtual environment.

The mobile applications were built using the framework presented in this thesis. Two important aspects were demonstrated. First of all the usage in the context of B2B applications instead of consumer applications. Secondly, it shows the plugability and exchangeability within the context management.

In this section, the development of the laboratory using the framework introduced in this work is presented. Firstly the scenario and outline the mobile applications is explained. Afterwards, the process of modelling a construction site and importing it into the system is shown. Finally, the applications' prototyping and the development of the virtual sensors are explained.

6.2.1. Scenario

The scenario is about discovering available machines and optimal order of tasks in a dynamic work plan. A static plan is opposed to a dynamic plan which always has the accurate data about every construction stage. While the dynamic planning obviously reduces the waiting times, it increases the information gathering effort needed to fill the databases with suitable data. The laboratory experiment is used to acquire the time for the following tasks:

- Finding a suitable next location to work if the currently scheduled tasks dependencies are not fulfilled (manually or using mobile technology).
- Walking through the whole construction site to gather the status of each craft, activity, or room.
- Entering the information about completed tasks.
- Finding idle physical resources.

The dynamic planning approach always (re-)optimizes the plan for the next task for the lowest overall waiting time using current data. The static approach does the same using work plans generated in advance.

6.2.2. Virtual Environment

To test the model in different scenarios, three different construction sites were modeled in a 3D editor. As a basis for the models, blueprints of three houses were used. One was of a small one-family house, one of a multi-family house, and one of a multilevel office building. Since the scenario is located in the interior work, the rooms are not equally completed but are in a different stage of construction. Those models are then exported into a Collada model. In this stage of the laboratory construction, it is essential to carefully adhere to the right sizes of the rooms and to follow a floor plan which can later be used in the simulated application.

Furthermore, all movable elements, such as machines or building materials, need to be designed. The models are then exported into the XML-based Collada file format, which is an open standard for 3D model exchange (Arnaud and Barnes 2006).

Test persons walk in a virtual environment recreating a construction site. Therefore, a 3D model viewer was built on a desktop PC. The model previously exported to the Collada format is then imported into an application which uses the JMonkeyEngine[3] for 3D vizualisation. Thus, it is converted into a more light-weight format which allows faster reading and smaller file size.

The application uses a first-person-shooter like navigation technique that allows the test subjects to move in a similar speed as on a real construction site. It tries to mimic human movement by having a similar height for the point of view and does only allow normal walking on normal stairs or ladders.

6.2.3. User Interface Design

To foster the creation of many variations, a UI framework was used which allows easy creation and manipulation of user interfaces. The best fit for this task was the UI toolkit shown in 4. Each screen visible to the user can easily be adapted to different devices with multiple display sizes, such as a phone or a PDA. To reduce the Java source code which has to be written for different experiments was one of the primary objectives when building the laboratory. So, most of

[3]http://www.jmonkeyengine.com

the application logic was omitted and the multi-screen dialog rules were used to simulate the workflow.

6.2.4. Searching Tools/Machines

Machine resources discovery is done using the service discovery component. Which machines are available is determined by the simulation. The results are then put out into an XML file containing machines' the location and the tasks they are utilizable for. These were used as the input for the service discovery server.

Using the framework on the mobile client, only the XUL designs and 20 lines of Java code, to get the results from the service discovery to feed them into the shown list, were needed.

6.2.5. Virtual Sensors

The virtual environment enables us to easily add context sensors which would be hard to build as a prototype in the real world to assess the benefit of having such a sensor.

Positioning System. To emulate the 802.11 indoor positioning approach (King, Haenselmann, Kopf, and Effelsberg 2007) used in SALSA project, a wrapper for JSR-179 which uses the positions of the virtual construction site and selectively reduces the precision by adding a random value which represents the real world precision was implemented. Two positioning systems were implemented. First of all, a GPS service. This service outputs user's the position within the simulated site with an accuracy similar to that of the GPS system. It only works if there is a direct line of sight to the sky (i.e. no ceiling above the worker) for at least the last 15 seconds.

The other one simulates a WLAN (802.11) based positioning system (King, Haenselmann, Kopf, and Effelsberg 2007). It has a good indoor accuracy, but only works within a predefined area. The accuracy is user-definable to be able to match the infrastructure of the site and the work to build the fingerprinting database.

Machine Status. If a machine is in use, it may be a valuable information for the planning of the next tasks. The "machine status" sensor sends the flag "in use" and the room the machine currently is in.

6.2.6. Collecting Data

To validate the model which predicts the benefit of the introduction of mobile applications in different shapes, all available data has to be captured. Therefore, the system captures the following data:

- Position and movement of the virtual person at any time;
- Screen of the mobile device;
- Each "Click" of the user.

Furthermore, the experiment including the user's reaction is filmed and later on synchronized with the simulated environment using a time stamp displayed on the user's screen.

6.2.7. Conclusions

Two important aspects were demonstrated. First of all the framework is easy to integrate into a virtual environment because of the loosely coupled sensor interfaces. The GPS components were simply exchanged to reflect the coordinates of the virtual environment.

Furthermore the system allows rapid prototyping of the application and the environment. All user interfaces were created without any programming knowledge and tailored for two different mobile terminals. Java code had only be written for the virtual sensors and to access the service discovery.

The whole system could be easily used on a real construction site just by exchanging the context sensors with real one, e.g. those from the consumer example above.

6.3. Summary

The two case studies showed the utility and feasibility of the framework in different scenarios. Especially the following features where demonstrated:

- *Context sensors:* The sensors could easily be plugged in and exchanged for a simulation in the second case. This shows the flexibility of the framework for any sensor. Several context data sources where implemented in the consumer application. The simulation environment added several more sensors for business data.

- *Context evaluation and service discovery* was used in different ways. The construction site application showed the next working place for workers using the context evaluation logic. In the consumer application the same component could be reused in several places: The service discovery in the first place used the same component, from the same codebase as the recommendation of the tourist guide or the restaurant guide without any changes. All these use cases were possible with the same set of operations.

- *Responsiveness* of the UI components was shown in the user tests of both applications.

- *Downloaded components and Virtual Machine:* Each scenario involved multiple components which were downloaded at runtime and executed within the virtual machine showed in chapter 5. None of the components was specially tailored for the VM and all together had a very large coverage of the Java VM opcodes (¿95 percent). All results were compared to the execution on a standard Java VM and no differences could be found.

- *Rapid development:* of services and user interfaces was extensively used to implement the variations needed in the simulation environment. The construction UI was designed by non-developers without problems.

7. Conclusions

This chapter offers a summary of the thesis. The major results and their scientific and practical implications are outlined. Finally, an outlook for possible future research will be provided.

7.1. Summary

Privacy in context-aware systems is an important issue. This work improves the state of the art in three distinct areas. First of all, a distributed discovery architecture is outlined in chapter 3. Using this method, it is possible to find suitable services for the current task of a user without sacrificing privacy. Service providers do only get degraded data which is not usable to track and analyze the user's behaviour. The full data is only used for processing on the client side. A prototypical implementation using a transformation from service descriptions is implemented and tested. Furthermore, the concept is applied to routing and navigation and is shown to offer a feasible trade-off between computation time, used bandwidth, and privacy protection.

Chapter 4 deals with automatic adaptation of user interfaces to varying devices and usage contexts. The adaptation mechanism needs a vast amount of context attributes and benefits of a large amount of usable information. By shifting the adaptation away from the application logic, the data can be used without leaking data to untrustworthy components. One of the biggest hurdles in developing mobile applications is the adaptation to many different devices having varying input and output capabilities. Using this UI toolkit, the adaption to these heterogeneous devices happens through rules specified within the UI description and does not require any changes to business logic.

The third part is about isolation of untrusted components and protection of data. A novel runtime environment, tracing data paths through components

at runtime and isolating sensitive information from outgoing network channels is built using a virtual machine prototype. Its usage fosters the download of untrusted components while still being able to use all sensitive data. These components are still able to use the network to communicate with the servers of the service provider, but will not be able to send private information.

All three parts are then put together into an easy to use framework for mobile application development. The suitability for this task is shown with the prototypical implementation of a typical consumer example application and a prototype for worker management at construction sites.

7.2. Contribution and Conclusions

It is known that privacy is an important aspect in mobile applications, especially when they are using context-aware features. Previous work was performed in two directions: First, data was classified as private or public and only the public attributes were used for decisions based on context and, second, different types of anonymization were used.

It is important to note that privacy aspects do need a holistic approach since discovering services under provisions for personal data are not reasonable if a service of an untrusted provider leaks the data at a later time. Therefore, the user interface adaption and service isolation has been analyzed and improved. Using these new concepts, a component downloaded from a service provider needs less data for the same amount of personalization and will be denied the right to send sensitive data to a server of the provider.

This thesis showed that an increased usage of context data, even with many, previously unknown service providers, is possible through a different distribution of the context processing between server and client or between components. Combining the chapters 3, 4, and 5 into a framework for mobile context-aware applications allows to create applications in short development time and an easy separation of design and logic. Developers do not need to care about the privacy aspect of a context-based search anymore, since the search can be used transparently, solely based on a transformation to the context rules outlined in chapter 3. Combined with existing approaches for anonymization, an even greater level of privacy protection is possible.

Together, these individual parts enable new and easy ways to develop context-aware applications. The implementation of demo-applications shows the feasability of the framework not only for the development of end-user applications, but also for easy and rapid prototyping of new user interfaces.

As more and more scientific publications in the last few years found an increased need for privacy aware architecture, this work shows a different direction for securing sensitive data. It can be combined with empirical results of relative importance of context attributes for the perceived protection of user privacy paired with anonymization to further decrease bandwith usage. While companies generally want to know more about the behaviour and preferences of the customers recent uproar about the data retention of search engines, advertising networks and social networks increases the importance of trustworthy privacy enhancing techniques to strengthen the customer relationship.

Another practical implication is the easier development of user interfaces for varying contexts and many heterogenous mobile devices. Using a widely known interface description language increases the number of available designers. This is paired with an easy to use rule language in order to adapt to different contexts.

In many economic sectors, there is a trend to alliances and there are emerging ecosystems around the mobile platforms. While some infrastructure providers do have trust problems with their open platform, some others do certify every application and, therefore, do not offer such a wide choice of applications. Using the framework presented in this thesis, this problem could be solved by opening the platform and still increasing the trust into applications.

7.3. Future Research

The scope of this work did not include the classification of context attributes. It is important for a user to have the ability to easily decide which attributes are sensitive or not. With an increasing number of context sensors and inferred context attributes, a new way to group them is needed to put them in the public/private group. This would require empirical research with a large number of real services and context sensors, not available today.

The impact of hiding an attribute on the bandwidth is currently based on historical data of one user. Some empirical data with a larger number of users and services would help to forecast the influence of each decision.

Traditional context-aware systems use servers which aggregate the context of a multitude of users to decide, based on the context of social groups (e.g. "what activities are possible with whom?"). Hence, some advanced P2P context sharing is needed to combine these group decisions with the aforementioned concepts. A first step in this direction is done by Leiner (Leiner 2006).

A. Context Serialized Format

The serialization of context data in all chapters uses the following schema:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="ContextBundle">
    <xsd:complexType>
      <xss:group ref="contextgroup" maxOccurs="unbounded"/>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>

<xsd:group name="contextgroup">
<xsd:choice>
<xsd:element name="Circle">
<xsd:complexType>
<xsd:sequence>
        <xsd:element ref="Point"/>
        <xsd:element name="Radius" xsd:decimal/>
</xsd:sequence>
</xsd:complexType>
</xsd:element>
        <xsd:element name="Point">
<xsd:complexType>
        <xsd:element name="Coordinate"/>
<xsd:complexType>
        <xsd:element name="Latitude" xsd:decimal/>
        <xsd:element name="Longitude" xsd:decimal/>
        <xsd:element name="Altitude" xsd:decimal/>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
        <xsd:element name="Polygon">
        <xsd:element ref="Coordinate">
</xsd:element>
<xsd:element name="Number">
```

```
        <xsd:complexType><xsd:sequence>
                <xsd:element name="Value" xsd:decimal/>
        </xsd:sequence></xsd:complexType>
</xsd:element>
<xsd:element name="NumberRange">
        <xsd:complexType>
                <xsd:element name="LowerBound" xsd:decimal/>
                <xsd:element name="UpperBound" xsd:decimal/>
        </xsd:complexType>
</xsd:element>
<xsd:element name="SingleDate">
        <xsd:complexType>
        <xsd:element name="Date">
        <xsd:complexType>
        <xsd:sequence>
                <xsd:element name="Hour" xsd:int/>
                <xsd:element name="Minute" xsd:int/>
                <xsd:element name="Second" xsd:int/>
        </xsd:sequence>
        </xsd:complexType>
</xsd:element>
<xsd:element name="Time">
        <xsd:complexType>
                <xsd:sequence>
                        <xsd:element name="Hour" xsd:int/>
                        <xsd:element name="Minute" xsd:int/>
                        <xsd:element name="Second" xsd:int/>
                </xsd:sequence>
        </xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
        <xsd:element ref="Time"/>
        <xsd:element name="DailyTimeInterval">
        <xsd:complexType>
                <xsd:sequence>
                        <xsd:element ref="Date"/>
                        <xsd:element type="Time" name="Start"/>
                        <xsd:element type="Time" name="Element"/>
                </xsd:sequence>
        </xsd:complexType>
</xsd:element>
</xsd:choice>
```

```
</xsd:group>
```

References

Ackerman, M., T. Darrell, and D. J. Weitzner (2001). Privacy in context. *Human-Computer Interaction 16*(2/4), 167–176.

Aleksy, M., C. Atkinson, P. Bostan, T. Butter, and M. Schader (2006). Interaction styles for service discovery in mobile business applications. In *9th International Workshop on Network-Based Information Systems (NBiS2006)*.

Aleksy, M. and R. Gitzel (2002). Design and implementation of a leasing service for corba-based applications. In *CW '02: Proceedings of the First International Symposium on Cyber Worlds (CW'02)*, Washington, DC, USA, pp. 0054. IEEE Computer Society.

Alexander, C., S. Ishikawa, M. Silverstein, et al. (1977). *A Pattern Language*. New York: Oxford University Press.

Anderson, R. (2008). *Security Engineering: A guide to building dependable distributed systems*. Wiley.

Andrade, R. and L. Logrippo (2000). A pattern language for mobility management. In *Andrade, R., Logrippo, L., et al, A Pattern Language for Mobility Management, Proc. of the Conference on Pattern Languages of Programming (PLoP), 2000.*

Arnaud, R. and M. Barnes (2006). *Collada: Sailing the Gulf of 3d Digital Content Creation*. AK Peters Ltd.

Barkhuus, L. and A. Dey (2003). Location-based services for mobile telephony: a study of users' privacy concerns. In *Proceedings of INTERACT 2003, 9th IFIP TC 13 International Conference on Human-Computer Interaction*.

Bauer, H., T. Reichardt, S. Exler, and E. Tranka (2007). Utility-based design of mobile ticketing applications–a conjoint-analytical approach. *International Journal of Mobile Communications 5*(4), 457–473.

Bauer, H., T. Reichardt, and A. Schuele (2005). User requirements for location-based services - an analysis on the basis of literature. *Wissenschaftliches Arbeitspapier Nr. W94, Institut fr Marktorientierte Unternehmensfhrung, Universitt Mannheim*.

Berghel, H. (2000). Identity theft, social security numbers, and the Web. *Communications of the ACM 43*(2), 17–21.

Bostan, P., T. Butter, and C. Atkinson (2008). SALSA a framework for context-sensitive service discovery in mobile commerce services. Technical report, University of Mannheim.

Broens, T., S. Pokraev, M. van Sinderen, J. Koolwaaij, and P. Costa (2004). Context-Aware, Ontology-Based Service Discovery. *Ambient Intelligence: Second European Symposium, EUSAI 2004, Eindhoven, The Netherlands, November 8-11, 2004: Proceedings*.

Buchholz, T., A. Küpper, and M. Schiffers (2003). Quality of context: What it is and why we need it. In *Workshop of the HP OpenView University Association 2003 (HPOVUA 2003)*.

Buschmann, F., R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal (1996). *Pattern-oriented software architecture: a system of patterns*. New York, NY, USA: John Wiley & Sons, Inc.

Butter, T., M. Aleksy, P. Bostan, and M. Schader (2007). Context-aware user interface framework for mobile applications. In *Proceedings of the 27th International Conference on Distributed Computing Systems - Workshops (ICDCS Workshops 2007)*, Toronto, Ontario, Canada.

Butter, T., S. Deibert, and F. Rothlauf (2006). Using private and public context - an approach for mobile discovery and search services. In T. Kirste, B. Koenig-Ries, K. Pousttchi, and K. Turowski (Eds.), *Mobile Informationssysteme - Potentiale, Hindernisse, Einsatz im Rahmen der Multikonferenz Wirtschaftsinformatik (MKWI) 2006*, pp. 144–155.

Butter, T., I. Duda, M. Aleksy, and M. Schader (2006). A framework for context-sensitive mobile applications. In *Proceedings of the IADIS International Conference e-Commerce 2006*, Barcelona, Spain, pp. 308–312.

Chen, G. and D. Kotz (2000, November). A survey of context-aware mobile computing research. Technical Report TR2000-381, Dartmouth College, Computer Science, Hanover, NH.

Cheverst, K., N. Davies, K. Mitchell, A. Friday, and C. Efstratiou (2000). Developing a context-aware electronic tourist guide: some issues and experiences. *Proceedings of the SIGCHI conference on Human factors in computing systems*, 17–24.

Clarke, E. (1999). *Model Checking*. MIT Press.

Corsaro, A., D. Schmidt, R. Klefstad, and C. ORyan (2002). Virtual component: A design pattern for memory-constrained embedded applications. In *Proc. 9th Conf. on Pattern Language of Programs (PLoP 2002)*, Monticello, USA.

Czajkowski, G. (2000). Application isolation in the Java Virtual Machine. *Proceedings of the 15th ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications*, 354–366.

D'Amorim, M. and K. Havelund (2005). Jeagle: A java runtime verification tool.

Deibert, S., A. Heinzl, and F. Rothlauf (2008). The impact logic of mobile technology usage on job production. In *Proceedings of the 14th Americas Conference on Information Systems*.

Dey, A. K. (2001). Understanding and using context. *Personal Ubiquitous Computing 5*(1), 4–7.

Dimitrakos, T., D. Golby, and P. Kearney (2004). Towards a Trust and Contract Management Framework for Dynamic Virtual Organisations. *Proceedings of the eChallenges Conference (eChallenges 2004)*.

Eisenstein, J., J. Vanderdonckt, and A. Puerta (2001). Applying model-based techniques to the development of UIs for mobile computers. *Proceedings of the 6th international conference on Intelligent user interfaces*, 69–76.

Fahy, P. and S. Clarke (2004). CASS–a middleware for mobile context-aware applications. In *Workshop on Context Awareness, MobiSys*. Citeseer.

Flanagan, D. (2002). *JavaScript: the definitive guide*. Éditions O'Reilly.

Floridi, L. (2003). *The Blackwell Guide to the Philosophy of Computing and Information*. Blackwell Publishing.

Fowler, M. (1997). *Analysis patterns: reusable objects models*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc.

Gamma, E., R. Helm, R. Johnson, and J. Vlissides (1995). *Design Patterns, Elements of Reusable Object-Oriented Software*. Professional Computing Series. Addison-Wesley.

Garfinkel, T., B. Pfaff, J. Chow, M. Rosenblum, and D. Boneh (2003). Terra: a virtual machine-based platform for trusted computing. *Proceedings of the nineteenth ACM symposium on Operating systems principles*, 193–206.

Garud, R. and A. Kumaraswamy (2002). Technological and Organizational Designs for Realizing Economies of Substitution. *The Strategic Management of Intellectual Capital and Organizational Knowledge*.

Gebauer, J., M. Shaw, and K. Zhao (2002). Assessing the Value of Emerging Technologies: The Case of Mobile Technologies to Enhance Business-to-Business Applications. *Proceedings of the 15 thBled Electronic Commerce Conference*.

Geihs, K. (2001). Middleware Challenges Ahead.

Goldberg, D. E. (1989). *Genetic algorithms in search, optimization, and machine learning*. Reading, MA: Addison-Wesley.

Golle, P. (2006). Revisiting the uniqueness of simple demographics in the us population. In *WPES '06: Proceedings of the 5th ACM workshop on Privacy in electronic society*, New York, NY, USA, pp. 77–80. ACM.

Gong, L. (1997). Java security: present and near future. *Micro, IEEE 17*(3), 14–19.

Gong, L., M. Mueller, H. Prafullchandra, and R. Schemers (1997). Going beyond the sandbox: An overview of the new security architecture in the java development kit 1.2. In *Proceedings of the USENIX Symposium*.

Gray, P. D. and D. Salber (2001). Modelling and using sensed context information in the design of interactive applications. In *EHCI*, pp. 317–336.

Gross, R. and A. Acquisti (2005). Information revelation and privacy in online social networks (the Facebook case). *Proceedings of the Workshop on Privacy in the Electronic Society*.

Grossmann, M., M. Bauer, N. Hönle, U.-P. Käppeler, D. Nicklas, and T. Schwarz (2005, März). Efficiently managing context information for large-scale scenarios. In *Proceedings of the 3rd IEEE Conference on Pervasive Computing and Communications: PerCom2005; Kauai Island, Hawaii, March 8-12, 2005*. IEEE Computer Society.

Guarino, N. (1998). *Formal ontology in information systems*. IOS Press.

Harrison, R. and M. Shackman (2007). *Symbian OS C++ for mobile phones*. Wiley.

Hart, P., N. Nilsson, and B. Raphael (1968). A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *Systems Science and Cybernetics, IEEE Transactions on 4*(2), 100–107.

Henning, M. and S. Vinoski (1999). *Advanced CORBA programming with C++*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc.

Henricksen, K., J. Indulska, and A. Rakotonirainy (2003). Generating Context Management Infrastructure from Context Models. *4th International Conference on Mobile Data Management (MDM), Industrial Track Proceedings*, 1–6.

Hevner, A. R., S. T. March, J. Park, and S. Ram (2004). Design science in information systems research. *MIS Quarterly 28*(1).

Hinze, A. and A. Voisard (2003). Locations- and time-based information delivery in tourism. In *SSTD*, pp. 489–507.

Ho, S. Y. and S. H. Kwok (2003, January). The attraction of personalized service for users in mobile commerce: An empirical study. *ACM SIGecom Exchanges 3*(4), 10–18.

Hull, R., B. Kumar, D. Lieuwen, P. Patel-Schneider, A. Sahuguet, S. Varadarajan, and A. Vyas (2004). Enabling context-aware and privacy-conscious user data sharing. *Mobile Data Management, 2004. Proceedings. 2004 IEEE International Conference on*, 187–198.

Inc., S. M. (2003). Jinitm architecture specificationversion 2.0.

Indulska, J., R. Robinson, A. Rakotonirainy, and K. Henricksen (2003). Experiences in Using CC/PP in Context-Aware Systems. *Mobile Data Management: 4th International Conference, Mdm 2003, Melbourne, Australia, January 21-24, 2003: Proceedings*.

Jacobson, I., G. Booch, and J. Rumbaugh (1999). *The unified software development process*. Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA.

Jain, P. (2001). Evictor. *Proceedings of 8 thPatterns Languages of Programs Conference (PloP 2001)*, 11–15.

Jain, P. and M. Kircher (2000a). Leasing. *Proceedings of 7 thPatterns Languages of Programs Conference (PloP 2000)*, 13–16.

Jain, P. and M. Kircher (2000b). Lookup Pattern. *Submitted to European Pattern Language of Programs conference, July*, 5–9.

Jain, P. and D. C. Schmidt (1997). Service configurator: A pattern for dynamic configuration of services. In *COOTS*, pp. 209–220.

Jarvenpaa, S. L., K. R. Lang, Y. Takeda, and V. K. Tuunainen (2003). Mobile commerce at crossroads. *Commun. ACM 46*(12), 41–44.

Kaasinen, E. (2003). User needs for location-aware mobile services. *Personal and Ubiquitous Computing 7*(1), 70–79.

Kafka, F. (1925). *Der Process*. Adamant Media Corporation.

Kao, T.-H., S.-P. Shen, S.-M. Yuan, and P.-W. Cheng (2003). An xml-based context-aware transformation framework for mobile execution environments. In X. Zhou, Y. Zhang, and M. E. Orlowska (Eds.), *APWeb*, Volume 2642 of *Lecture Notes in Computer Science*, pp. 132–143. Springer.

Kaplan, E. and C. Hegarty (2005). Understanding GPS: Principles and Applications Second Edition. *Artech House 680*.

King, T., T. Haenselmann, and W. Effelsberg (2007, September). Deployment, Calibration, and Measurement Factors for Position Errors in 802.11-based Indoor Positioning Systems. In J. Hightower, B. Schiele, and T. Strang (Eds.), *Proceedings of the Third International Symposium on Location- and Context-Awareness (LoCA)*, Volume 4718 of *Lecture Notes in Computer Science*, Oberpfaffenhofen, Germany, pp. 17–34. Springer.

King, T., T. Haenselmann, S. Kopf, and W. Effelsberg (2007, 03). Overhearing the Wireless Interface for 802.11-based Positioning Systems. In *Proceedings of the Fifth IEEE International Conference on Pervasive Computing and Communications*, pp. 145–150.

Kircher, M. (2001). Lazy Acquisition. *Proceedings of EuroPlop*.

Köchy, V. (2008). *Evaluierung von Plattformen und Frameworks für die mobile Anwendungsentwicklung*. GRIN Verlag.

Kuck, J. and F. Reichartz (2005). A collaborative and feature-based approach to Context-Sensitive Service Discovery.

Lee, S., S. Ko, and G. Fox (2003). Adapting content for mobile devices in heterogeneous collaboration environments. *Proceedings of the International Conference on Wireless Networks 2003*, 23–26.

Lee, Y. E. and I. Benbasat (2003). Interface design for mobile commerce. *Commun. ACM 46*(12), 48–52.

Leiner, C. (2006). *Privacy-Aware Context Data Sharing on Mobile Devices via SIP/SIMPLE*. Diploma Thesis. University of Mannheim.

Lin, T. (2006). Chinese Wall Security Policy Model: Granular Computing. *Web and Information Security*, 196.

Lindholm, T. and F. Yellin (1999). *The Java(tm) Virtual Machine Specification - Second Edition*. SUN.

Lopes, C. and W. Hursch (1995). Separation of Concerns. *College of Computer Science, Northeastern University, Boston, February 1995*.

Lutz, M. (2006). *Programming Python*. O'Reilly Media, Inc.

March, S. and G. Smith (1995). Design and natural science research on information technology. *Decision Support Systems 15*(4), 251–266.

May, H. and G. Hearn (2005). The mobile phone as media. *International Journal of Cultural Studies 8*(2), 195.

Mitchem, T., R. Lu, and R. OBrien (1997). Using kernel hypervisors to secure applications. In *Proceedings of the Annual Computer Security Applications Conference*.

Mitrovic, N. and E. Mena (2002). Adaptive user interface for mobile devices. In P. Forbrig, Q. Limbourg, B. Urban, and J. Vanderdonckt (Eds.), *DSV-IS*, Volume 2545 of *Lecture Notes in Computer Science*, pp. 29–43. Springer.

Moore, G. E. (1965, April). Cramming more components onto integrated circuits. *Electronics 38*.

Mostefaoui, S. and B. Hirsbrunner (2004). Context Aware Service Provisioning. *Proceedings of the IEEE/ACS International Conference on Pervasive Services (ICPS 2004), IEEE*, 71–80.

Muchow, J. (2001). *Core J2ME technology and MIDP*. Prentice Hall PTR Upper Saddle River, NJ, USA.

Nichols, J., Z. Hua, and J. Barton (2008). Highlight: a system for creating and deploying mobile web applications. In *Proceedings of the 21st annual ACM symposium on User interface software and technology*, pp. 249–258. ACM New York, NY, USA.

OMG (1995). *The Common Object Request Broker: Architecture and Specification*.

Orwell, G. (1949). *Nineteen Eighty-Four*.

Ousterhout, J. (1994). *Tcl and the Tk toolkit*. Addison-Wesley Reading, Mass.

Pärssinen, J., K. Teemu, and P. Eronen (2005). Pattern Language for Service Discovery. *Europlop - Ninth european conference on pattern languages of programs, July 7-11 2004, Irsee, Germany 2004*.

Pokraev, S., J. Koolwaaij, M. van Setten, T. Broens, P. Costa, M. Wibbels, P. Ebben, and P. Strating (2005). Service platform for rapid development and deployment of context-aware, mobile applications. *Web Services, 2005. ICWS 2005. Proceedings. 2005 IEEE International Conference on*, 646.

Rayport, J. and B. Jaworski (2001). *E-commerce*. McGraw-Hill/Irwin MarketspaceU Boston.

Razzaque, M., S. Dobson, and P. Nixon (2005). Categorization and Modelling of Quality in Context Information. *Proceedings of the IJCAI 2005 Workshop on AI and Autonomic Communications*.

Römer, K., T. Schoch, F. Mattern, and T. Düben-dorfer (2003). Smart identification frameworks for ubiquitous computing applications. In *PerCom*, pp. 253–. IEEE Computer Society.

Rubin, A. and D. Geer Jr (1998). Mobile code security. *Internet Computing, IEEE 2*(6), 30–34.

Satyanarayanan, M. (1996). *Fundamental challenges in mobile computing*. ACM Press New York, NY, USA.

Schilit, B., M. Theimer, and B. Welch (1993). Customizing Mobile Applications. *Proceedings USENIX Symposiumon Mobile & Location-indendent Computing*, 2.

Schiller, J. H. (2004). *Location-Based Services*. Morgan Kaufmann.

Schmidt, A., M. Beigl, and H.-W. Gellersen (1999). There is more to context than location. *Computers & Graphics 23*(6), 893–901.

Schmidt, D. C., H. Rohnert, M. Stal, and D. Schultz (2000). *Pattern-Oriented Software Architecture: Patterns for Concurrent and Networked Objects*. John Wiley & Sons, Inc.

Schöning, U. (2000). *Logik für Informatiker*. BI Wissenschaftsverlag.

Shehzad, A., H. Q. Ngo, K. A. Pham, and S. Lee (2004). Formal modeling in context aware systems. In *Workshop on Modeling and Retrieval of Context, CEUR*.

Slominski, A. (2007). XMLPullParser. Website: http://www.xmlpull.org/.

Solove, D. (2007). "i've got nothing to hide" and other misunderstandings of privacy. *San Diego Law Review 44*.

Stal, M. (99). "des knaben wunderhorn", kommentare des fachbeirats komponenten-forum. *OBJEKTspektrum 1*, 18–20.

Strang, T. and C. Linnhoff-Popien (2004). A Context Modeling Survey. *Workshop on Advanced Context Modelling, Reasoning and Management, UbiComp*, 34–41.

Stremersch, S. and G. Tellis (2002). Strategic Bundling of Products and Prices: A New Synthesis for Marketing. *Journal of Marketing 66*(1), 55–72.

Sun (2004). J2ME Connected Limited Device Configuration.

Sun (2005). JSR218: J2ME Connected Device Configuration.

Sweeney, L. (2000). Uniqueness of simple demographics in the us population. *LIDAP-WP4. Carnegie Mellon University, Laboratory for International Data Privacy, Pittsburgh, PA*.

Szekely, P., P. Luo, and R. Neches (1993). Beyond interface builders: model-based interface tools. *Proceedings of the SIGCHI conference on Human factors in computing systems*, 383–390.

Tatli, E. I., D. Stegemann, and S. Lucks (2005). Security challenges of location-aware mobile business.

Tatli, E. I., D. Stegemann, and S. Lucks (2006). Dynamic mobile anonymity with mixing. Technical report, University of Mannheim.

Tolia, N., D. Andersen, and M. Satyanarayanan (2006). Quantifying Interactive User Experience on Thin Clients. *COMPUTER*, 46–52.

Traynor, C. (2008). *Swing and AWT*. Newport House Books, LLC Whitehouse Station, NJ, USA, USA.

Tsalgatidou, A., J. Veijalainen, and E. Pitoura (2000). Challenges in Mobile Electronic Commerce. *Proc. of 3rd International Conference on Innovation through E-Commerce, UK*.

van Setten, M., S. Pokraev, and J. Koolwaaij (2004). Context-aware recommendations in the mobile tourist application compass. In *AH*, Volume 3137 of *Lecture Notes in Computer Science*, pp. 235–244. Springer.

Varian, H. (2002). Economic aspects of personal privacy. *Cyber Policy and Economics in an Internet Age*.

Varshavsky, A., M. Chen, E. de Lara, J. Froehlich, D. Haehnel, J. Hightower, A. LaMarca, F. Potter, T. Sohn, K. Tang, , and I. Smith (2006, April). Are gsm phones the solution for localization? In *IEEE Workshop on Mobile Computing Systems and Applications*.

Varshney, U., R. Vetter, and R. Kalakota (Oct 2000). Mobile commerce: a new frontier. *Computer 33*(10), 32–38.

Vogel, O. (2001). Service abstraction layer. In *Proceedings of the 6th European Conference on Pattern Languages of Programs (EuroPLoP 2001)*, Irsee, Germany.

Vlter, M. (2001). Server-side components — a pattern language. In *Proceedings of the 6th European Conference on Pattern Languages of Programs (EuroPLoP 2001)*, Irsee, Germany.

Vlter, M., M. Kircher, and U. Zdun (2000). Object-oriented remoting – basic infrastructure patterns. In *Proceedings of the 5th European Conference on Pattern Languages of Programs (EuroPLoP 2000)*, Irsee, Germany.

W3C (2005). Document Object Model (DOM) Level 3 Events Specification. http://ww.w3.org/TR/DOM-Level-3-Events/.

Wang, X., D. Zhang, T. Gu, and H. Pung (2004). Ontology based context modeling and reasoning using OWL. In *Pervasive Computing and Communications Workshops, 2004. Proceedings of the Second IEEE Annual Conference on*, pp. 18–22.

Welch, L. R., T. Marinucci, M. W. Masters, and P. V. Werme (2002). Dynamic resource management architecture patterns. In *Proceedings of the 9th Conference on Pattern Languages of Programs (PLoP 2002)*, Monticello, USA.

Westin, A. (1967). *Privacy and Freedom*. Atheneum New York.

Wind, R., C. Jensen, and K. Torp (2007). Windows Mobile Programming. *Mobile Phone Programming and Its Application to Wireless Networking 1*, 207.

Wirfs-Brock, R. J. and R. E. Johnson (1990). Surveying current research in object-oriented design. *Commun. ACM 33*(9), 104–124.

XUL Tutorial (2006). http://www.xulplanet.com/tutorials/xultu/.

Ye, J. and J. Herbert (2004). Interface Tailoring for Mobile Computing Devices. In *User-Centered Interaction Paradigms for Universal Access in the Information Society: 8th ERCIM Workshop on User Interfaces for All, Vienna, Austria, June 28-29, 2004: Revised Selected Papers*. Springer.

Zdziarski, J. (2008). *Iphone open application development*. O'Reilly.

Zhang, X. (1997). Secure code distribution. *Computer 30*(6), 76–79.

Lebenslauf

Persönliches

Name	Thomas Butter
Geburtsort	Heidelberg
Geburtstag	
Staatsangehörigkeit	deutsch
E-Mail	

Ausbildung

09 / 1989 – 06 / 1998	Peter-Petersen-Gymnasium, Mannheim (Abschluss Abitur)
10 / 1999 – 04 / 2004	Studium, Universität Mannheim (Abschluss Diplom Wirtschaftsinformatiker)
08 / 2004 – 03 / 2008	Wissenschaftlicher Mitarbeiter, Lehrstuhl für ABWL und Wirtschaftsinformatik (Prof. Dr. Armin Heinzl) und Lehrstuhl für Wirtschaftsinformatik III (Prof. Dr. Martin Schader), Universität Mannheim