Mobile Information Systems 4 (2008) 105–117
IOS Press

Architecture for the development of context-sensitive mobile applications

Markus Aleksy*, Thomas Butter and Martin Schader
University of Mannheim, Mannheim, Germany
E-mail: {aleksy, butter, schader}@wifo3.uni-mannheim.de

Abstract. Recent advances in the development of mobile terminals and the appropriate communication infrastructures have the consequence that new kinds of applications arise. This trend leads to more and more complex mobile client applications as well. In this paper, we present a generic architecture, which can be used for the development of context-sensitive mobile applications.

Keywords: Context-sensitive applications, mobile computing, adaptable applications, software engineering

1. Introduction

The increasing acceptance of mobile terminals together with the progressive development of handheld devices and the simultaneous improvement in the infrastructure of wireless communication play a more and more important role. Applications, which are executed on mobile terminals often require both a simultaneous interaction with other users of mobile terminals as well as with fixed or location-dependent services. Although this development is beneficial from the view of the end-user, it confronts application developers with the problem that they have to manage the growing complexity of mobile applications.

A main characteristic of mobile applications is their dynamics. It is therefore important that during development of a mobile application, its dynamic adaptability must be taken into account as one of the most important design criteria. Next to that, other criteria, such as the wishes and concerns of the users as well as the technical restrictions of mobile terminals have to be considered, simultaneously.

In this paper, we present an architecture for the development of mobile context-sensitive applications, which fulfills a variety of the existing requirements and can therefore simplify the realization of applications of this type.

2. Overview

The University of Mannheim's Mobile Business Research Group (see http://m-business.uni-mannheim.de) is engaged in the realization of a generic platform for the implementation of location-related mobile business applications. The main emphasis is on transactions in which the current location

*Corresponding author: Markus Aleksy, Department of Information Systems, University of Mannheim, Schloss (L 5,5), 68131 Mannheim, Germany. E-mail: aleksy@uni-mannheim.de.

of the mobile terminal plays an important role. The project is subdivided into seven partial projects, which deal with various aspects of mobile applications, such as requirements capturing, generic components, position ontologies, service-oriented software architecture, or security aspects. In this paper, we concentrate on the specification of an architecture, which represents the fundamentals for the development of adaptable mobile applications. We concentrate mainly on the representation of the notable features of the client environment, such as the restrictions regarding the available resources, the processing of the current context information, as well as the user's preferences, and we describe how these aspects were taken into account in the context of our architecture.

3. Challenges to the development of context-sensitive mobile applications

A mobile application must fulfill a variety of different requirements. One can differentiate between technical challenges resulting from the restrictions of the mobile terminals as well as the requirements of the users.

3.1. Restrictions of the mobile terminals

Differing from workstations, PCs, and laptop computers, today's mobile terminals, such as PDAs or mobile phones, show a number of restrictions. Some of these are listed below:

- *Computational Aspects*
  Criteria like the computer power of the device or the available storage capacity fall into this category.
- *Input/Output Capabilities*
  Many of the mobile terminals have a strongly limited functionality regarding their input/output capabilities.
- *Communication*
  Communication on wireless networks represents another restriction. Next to the reduced bandwidth in comparison with wired systems, mobile applications must also manage higher latency periods. Furthermore, the costs for the use of a certain bandwidth play an important role since they often develop disproportionately high.
- *Power Supply*
  The energy supply of a mobile terminal is determined by the capacity of the accumulator or the battery.

Two further points of view are also important: Mastering the heterogeneity of mobile terminals as well as interoperability with an enterprise's existing information systems. While the first aspect improves the portability of the developed mobile applications, the second criterion helps business processes to profit smoothly from the advantages of mobile context-sensitive applications.

3.2. User requirements for location-based services

Besides the technical challenges mentioned before, the requirements of the users, who should as best work with the mobile applications form another hurdle. According to the studies carried out by Bauer, Reichardt, and Schüle [5,6], the following features are of special importance:

- *Customization options for the display of the service categories*
  The display of the service categories, which were in advance defined in the user profile is the most preferential solution. Display of the service categories, which were most frequently used follows directly after that.
- *Providing information on the available services*
  Here, the users wish that a detailed description of the services can be found on the Internet, and that, in addition, a short description is presented on the mobile terminal.
- *Display of available services*
  The services already used by the user should be displayed first.

Storage of the user profiles is another interesting feature. Most users prefer storage of their profile on the mobile terminal to storage on a central server.

The restraints and concerns of the users regarding the use of mobile applications represent another challenge. Here, the following aspects were stated among others: low data security, supervision, complexity, loss of control, and costs. The results of the analysis of the user interviews were taken into account during conception of the prototypical implementation of applications, which were realized based on the architecture introduced here.

4. Architecture for the development of context-sensitive mobile applications

An architecture for the development of context-sensitive mobile applications must fulfill a variety of different tasks at the same time. Next to the recording and management of different context information and the efficient management of resources of the mobile terminal, the user's search for services must be carried through very flexibly. Found results must be presented suitably to users, with consideration of the specific conditions of the mobile terminal. Furthermore, the communication means should be designed flexible, in order to enable effortless information exchange with existing information systems.

To do justice to these criteria, the architecture developed by us consists of the following components:

- *Context Manager*
  This component is responsible for the recording and management of the current values of the individual context sensors. In addition, it provides an interface, which can be used by the Service Discovery Architecture as well as other context-sensitive applications to obtain the current context information.
- *Component Manager*
  The Component Manager manages the life cycle of any application component, which is used on the mobile terminal. It is supported by a Class Loader component. This permits that application components, which provide a dedicated business logic or offer a user interface specifically adapted to the just requested service, are loaded dynamically.
- *Service Discovery Architecture*
  This part of the overall architecture is responsible for discovery of services. It completes user queries by the current context information. If the result of the query should require installation of a new component, then it falls back on the functionality offered by the Component Manager.
- *Adaptable User Interface Framework*
  This part of our architecture is helpful to be overcome the heterogeneity of the different types of terminals. With the help of this framework, it is possible to realize GUIs, which are device-independent as well as user-friendly.

– *Generic Communication Framework*
  To integrate mobile context-sensitive applications into existing business applications seamlessly, the provided communication infrastructure must be highly flexible. In addition to default support of protocols, which are often used in the area of Enterprise Computing, e.g., SOAP [14] or the Internet Inter-ORB Protocol (IIOP) [17], a generic infrastructure, which enables the usage of additional communication protocols, should be provided.

In the following subsections, the individual elements of our architecture are discussed in more detail.

4.1. Context manager

The realization of context-sensitive applications implies a flexible context processing. The first step consists in the recording of the single chunks of context information. Different context sensors are responsible for this task. Scheer [18] distinguishes four kinds of context information:

– *Environmental Context*
  This type of context information captures the environmental conditions of the current site, such as the weather or the existing lighting conditions.
– *Activity Context*
  This category describes the present activity of the user. Both professional as well as private activities, such as traveling or shopping may be captured, here.
– *Temporal Context*
  Recording of the current temporal conditions represents the third kind of context information. Besides the current clock time also other temporal factors, e.g., the season are to be found in this category.
– *Personal Context*
  This context includes individual preferences and characteristics of the user. The range covered here is very wide and must take into account a variety of factors. Some of these aspects may be mapped very easily, such as smoker vs. non-smoker, others may be more complex, for example the health status or interests and hobbies of a user.

Some context information is acquired by the mobile terminal itself – if the device has the corresponding technical prerequisites. Examples are the current position, line of sight, a calendar, etc. Since the mobile terminals currently offered on the market only have a relatively low number of context sensors at their disposal, some part of the context information is recorded by context sensors on the server side. Examples are the current weather conditions or the season.

When providing and using context information, which is included by the context sensors of the mobile terminal itself, it is important to deal efficiently with the resources of the device. Recording or updating context information too often might have the consequence that the computer power and the related energy consumption are unnecessarily wasted. To counterbalance this effect, our architecture supports two kinds of context sensors: Push sensors and Pull sensors. The first kind of context sensors scan their environment permanently and inform the Context Manager actively if any change has occurred. The recording of the current position by means of GPS is carried out in this way, for example. For a certain period of time, the current information is stored intermediately by the Context Manager. This way, repeated queries can be answered directly by it, without having to contact the context sensors anew. Pull sensors, on the other hand, act passively: they wait until the Context Manager questions them on their current status. Both, the intermediate storage of the context information and the occasional activation of Pull sensors help conserving the resources of a mobile terminal.

The number and the types of the context sensors on a mobile device can vary considerably. To circumvent the storage restrictions of a mobile terminal, only certain context sensors are installed by default. Typically, localization, user profile, time planner, etc. are included. When required, further context sensors can be downloaded and installed via an existing network connection. This task is dealt with by the Component Manager and Class Loader components (see Section 3.2).

In order to take into account the restraints and concerns of the users (see Section 2.2), our framework supports the possibility to declare all context attributes "public", "blurred", or "private". The values of an attribute declared "public" by a user are implicitly added to a service query. As opposed to this, "private" attribute values are never inserted in a service discovery query. Context information labeled "blurred" is first fuzzyfied before it is integrated into the service query. For example, if users would not like their current location to be disclosed, they can indicate that with a corresponding "blurred" declaration. The framework then calculates a random offset (e.g., +500 meters), which is added to the current location.

To provide appropriate support for the developers of mobile applications represented another challenge – besides the objective of satisfying user requirements. The Context Manager therefore supplies an API, which covers the basic operations for context processing. Examples for these are operations that record context information, operations that serialize service queries, or operations that analyze context situations with regard to similarity of attribute values. A complete description of the employed methods can be found in [10].

4.2. Component manager

Mobile applications, especially when they have to be executed in a location-aware context, need to be highly flexible. The reason is that context information, e.g., the current location of the user or the device, decides on the utilizable services. Such applications, therefore, must have two specific features: the ability of ad-hoc configuration and the ability to efficiently discover, add, manage, and use services. Although one could think that the aspect of the dynamic adaptability of a mobile client application concerns only the client side, in the majority of cases, this is not so (see Section 3.3). The reason is that flexibility with regard to detection, usage, and management of services is not limited to the client application alone but also requires a flexible infrastructure on the service side as well. Another problem, which must be solved is how to deal efficiently with the resources of the mobile client. As already explained in Subsection 2.1, mobile terminals have a rather limited storage capacity.

To solve this problem, our architecture contains a Class Loader, which permits reloading components dynamically. The loader is complemented by the Component Manager, which supervises the life cycle of the reloaded component. The life cycle of any reloaded component is characterized by the following phases:

- *Load*
  The framework notices that a component must be reloaded and loads it.
- *Install*
  Once the download process is completed successfully, the component is installed on the mobile terminal.
- *Register*
  After successful installation, the component is registered and subsequently is available to all applications residing on the mobile terminal.
- *Remove*
  If the functionality offered by the component is not needed any longer, the component is garbage collected, automatically.

The reloaded parts of the program can consist in context sensors or in components, which realize a service-specific GUI, or also components implementing a specific business logic.

Since downloaded components can access private context data (see Section 3.1), it must be made sure that components originating from not trustworthy sources cannot send these data via the net. Therefore, our architecture provides the possibility of redeploying components individually into an own Java Virtual Machine (JVM), which interprets classes within the device's JVM. A JVM is able to monitor the use of data. Such a component then can use private context information to adapt to the current situation of the user, but it cannot send such data via the net. The JVM earmarks private data and tracks, which data is generated from it. Should a downloaded component send such data via the net, the action is stopped and an exception is thrown. As usual, "trusted code" is executed in the device's JVM and therefore runs with higher performance. Since our framework already contains many trustworthy components, e.g., the GUI framework, only minor losses of performance occur.

Through the possibility of reloading context sensors or components, on the one hand, the required storage capacity can be reduced; on the other hand the context-sensitive mobile application be designed very flexibly.

4.3. Context-driven service discovery architecture

Mobile users have an interest in obtaining access to the services which, in their given environment, best meet their needs. From the perspective of the user, this can be characterized as personalization. Usability and personalization are even more significant for mobile applications than for traditional web-based services [20]. Discovering useful the most suitable services for a given context, services require a lot of information describing the user's current situation. Even if it were possible, given the limited input capabilities of today's mobile devices, enter all this information by hand would be extremely cumbersome. To overcome these restrictions and avoid intensive user-application interaction we enrich the user input with information gathered from context sensors. Using this additional information in the service discovery process considerably enhances the relevance of the search results in relation to the user's current situation [11].

The service-oriented architecture for the context-driven service discovery contains the following components:

- The Mobile Client component is a client application, built using the client framework developed by the Mobile Business Research Group (see http://m-business.uni-mannheim.de). This SALSA (Software Architecture for Location-Specific trAnsactions) framework is running on the mobile devices; it is responsible for mediating the communication with service brokers and service providers, and handles most of the context sensing. The mobile client offers a generic interface to the mobile user to query services and for displaying query results in the form of service descriptions. The client application adds implicit client-detected context information to the user's service discovery request if available and explicitly allowed by the user.
- The Service Discovery Service (SDS) is a service broker that stores detailed descriptions of services that have been explicitly registered in its service repository. The SDS is able to retrieve services matching the users explicitly specified requirements augmented with context information from the mobile client and additional context provisioning services. In every architectural configuration there is always one first-level SDS so-called Universal Service Discovery Service (USDS). The USDS is configured as the bootstrapping SDS for the mobile client and is the first component queried for suitable services. Since the SDS technology is based on the composite pattern [12], the services

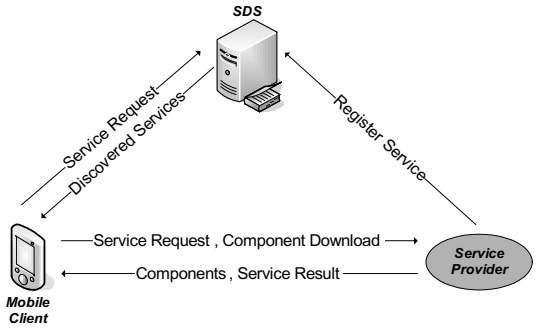

Fig. 1. Basic configuration.

returned by the USDS can be other SDSs that act as further service brokers, e.g. a gastronomy guide where restaurants are registered as services. SDS's can thus be nested in arbitrary ways to best exploit existing repositories or service providers.

– The Service Orchestration Engine (SOE) is a control component that coordinates the interactions with multiple, lower-level SDSs in a nested architecture. SOE's usually exists within SDSs but they can also be contained by client applications. A SOE coordinates service invocations and returns their merged results according to the Façade Pattern [12]. Additionally the SOE is able to invoke any service as a proxy for other components.
– Service Providers offer a service which may be used by any service requestor. A service may be an electronic service, e.g. a web service based gastronomy guide or a non-electronic service, a café. Service providers may adittionally offer lower-level service brokers that offer a certain kind of service using SDS technology. An SDS is also a form of service provider.
– Context Provisioning Services (CPS) deliver implicit contextual information or they derive new contextual information from existing information for the service discovery process. These services can be integrated into the mobile client application or the SDS.

The basic configuration of the presented service discovery architecture follows the paradigm of service-oriented architectures represented by a triangle between the service requestor (mobile client), service broker and service provider (see Fig. 1). This configuration is described by the Lookup pattern [15].

Service providers are registered at an SDS. The mobile client sends a service request augmented with available context information from client-side CPSs to the SDS which can add further context from server-side CPSs that are available. The SDS returns the most suitable service descriptions corresponding to the request and context from its repository to the mobile client. Finally, the mobile client directly invokes one of the returned service providers using the service descriptions from the SDS. If the chosen service is an electronic service it is invoked over the Internet and the user receives the delivered value.

Using the described SDS technology different configurations can be configured in which an SDS may act as a proxy to other services, return service descriptions directly or return pointers to other SDSs which are better able to deal with the given client query. The four main configurations are referred to as: User-Managed Linear Configuration, Client-Managed Linear Configuration, Client-Mediated Linear Configuration, and Server-Managed Hierarchical Configuration. For our project, which focuses on scenarios in which the user wants to find a previously unknown service in an ad hoc way depending on the current situation, we have identified the optimal configuration as the Client-Mediated Configuration (see Fig. 2).

Other approaches for the design of a similar Service Discovery Architecture can be found in [1,16]. A detailed description of the approach introduced here can be found in [2].

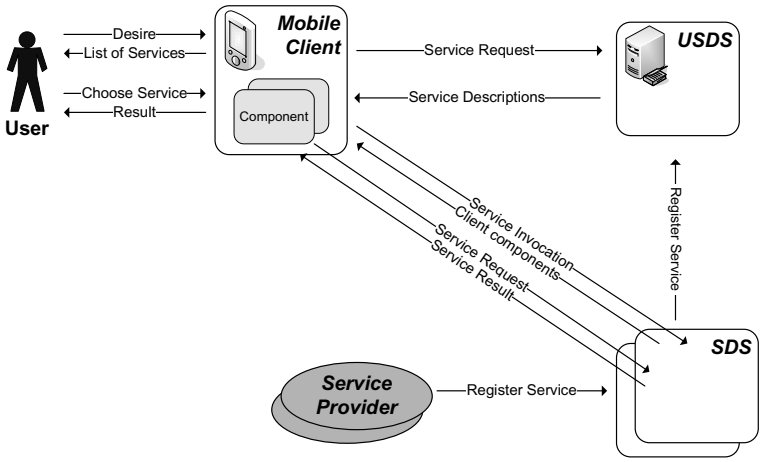

Fig. 2. Client-mediated linear configuration.

4.4. Adaptable user interface framework

The realization of a framework for the flexible design of user interfaces for mobile context-sensitive applications is affected by different objectives. Besides overcoming the heterogeneity of the different mobile devices, which is reflected in restricted input-/output capabilities, e.g., limited by the size of the display (see Section 2.1), also context-specific information must be presented in a user-friendly way (see Section 2.2). Furthermore, other aspects, such as adjustment of the user interface to the current user context, automatic adjustment of the presentation according to the available resources of the mobile device, or possibilities of simple GUI implementation, have to be considered. The first aspect also implies the support of the various existing GUI libraries. Since we have realized the architecture described here prototypically in Java, the following explanations refer to this programming language and the libraries available for Java.

The first GUI framework for Java applications was the Abstract Window Toolkit (AWT), which – since Java version 1.2 – was replaced by the Swing framework. AWT nevertheless is part of every J2SE as well as every Java ME CDC Personal Profile distribution. The javax.microedition.lcdui package is used for lightweight mobile terminals, such as mobile phones or low-end PDAs. It was introduced in the context of the Mobile Information Device Profile (MIDP) / CLDC [19] platform and supports only relatively few elements of a typical GUI.

The improvements provided by Swing framework simplify the programming of GUIs and, with the use of platform-specific Look & Feel, support a better integration of Java-based applications. However, the use of GUIs, which are based on Swing, as opposed to AWT, requires substantially more resources, particularly more storage. Swing technology, therefore, is not yet practicable in the area of the resource-weak mobile terminals but only with laptop computers and Tablet PCs. Bridging of the different functionalities offered by CLDC/MIDP and CDC/Personal Profile represents an additional challenge.

We have therefore decided to choose a different approach. It relies on the XML User Interface Language (XUL) [21], which is used in the context of different open source projects. The main feature of this approach is based on the separation between the structure, the actions/behaviors as well as the representation of a user interface. The GUI's structure is described in XUL, the actions and behaviors are implemented by Java components, and display information is declared in the form of Cascading Style Sheets (CSS) [7].

Fig. 3. The application on a low resolution, low contrast screen.

Fig. 4. The application on a high end PDA.

The automatic adjustment of the user interface to the current context, the user-friendly and efficient representation of the available information as well as the restrictive usage of the terminal's available resources reflect the concepts, which have guided our design. The representation of the detected location-related services is carried out bearing the user requirements of Section 2.2 in mind. To improve the quality of the representation, the framework measures the time necessary to display the GUI. If this should be recognized as too slow, then different measures are taken. Next to deactivation of animations and purely visual effects, also effects, which necessitate hardware just not available also will be switched off automatically. Effects such as, e.g., "hover" fall into this category. Furthermore, the framework permits that pictures are replaced by descriptive text if the display format or the available resources do not suffice for a graphical presentation. Figures 3 and 4 show the different representations of the same GUI on devices with differing performances.

The UML diagram in Fig. 5 shows three typical scenarios with interactions between the user or the application, the Context Manager component, and the GUI components. The first scenario describes a GUI creation, the reaction of the frameworks to a change of context is represented in the second scenario, while the third scenario is representing a user's action.

To meet the challenge of heterogeneity, or, more precisely, to adjust to the available computing power, the framework supports two different approaches: the GUI can be either interpreted or compiled. While the first case is bringing about the great advantage that alterations in the user interface do not imply that

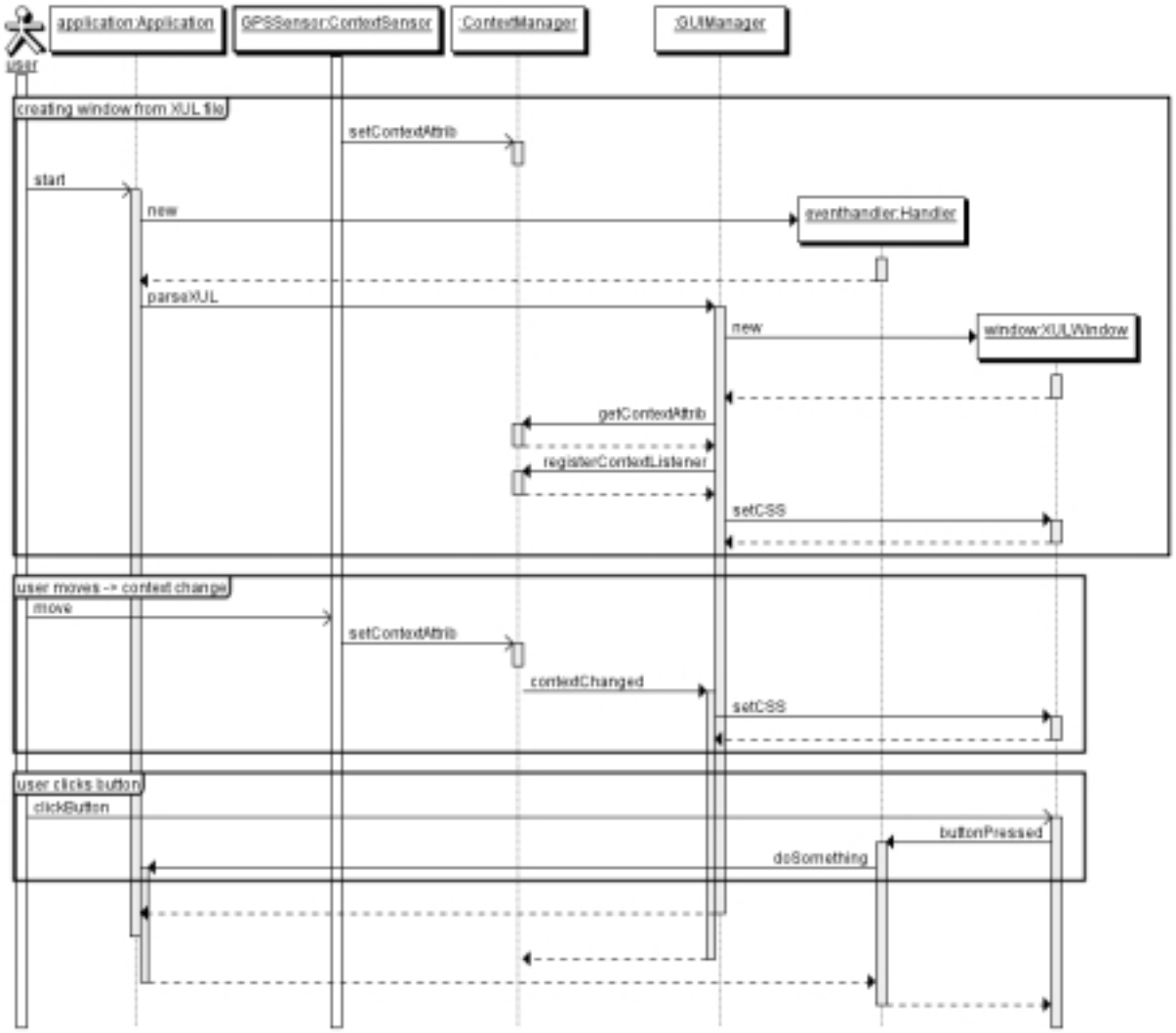

Fig. 5. Example scenarios of the GUI framework at work.

the complete GUI has to be recompiled, the second alternative permits a more efficient construction of the GUI. Figure 6 shows the required execution times when employing the interpreted approach (continous line) and the compiled approach (broken line), respectively. The compiled XUL files are tested against a J9 VM on a Sharp Zaurus using the Personal Profile version of the XUL rendering backend. A UI from a 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. Compilation improves performance by a factor of 7 for 5 elements, which is a quite common ratio for mobile user interfaces and a factor of still 5 for 20 elements; here, the absolute time for UI creation is decreased by 110 msec, which makes noticeable difference even on this relatively fast PDA. On slower PDAs or mobile phones, the difference in execution time is even more important.

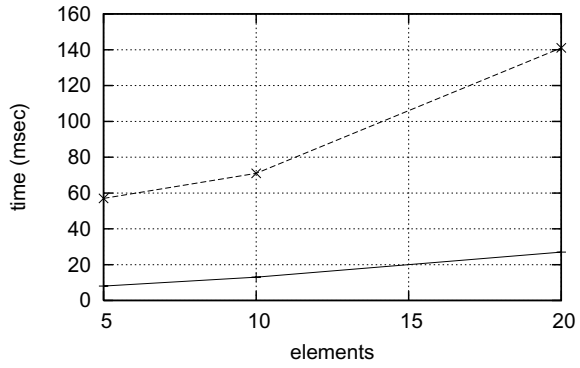

Fig. 6. Performance measurements.

4.5. Generic communication framework

Although the introduced elements of our architecture both take into account the requirements and the needs of the users regarding the use of mobile context-sensitive applications and take into account the restrictions of mobile terminals, these aspects do not suffice to realize context-sensitive mobile business applications. The reason for that is the necessity that the mobile application can communicate with an Enterprise Information System. Only this way, existing business processes can benefit from the use of mobile context-sensitive applications. It is therefore necessary that communication protocols well-established in Enterprise Computing such as SOAP [14] for the Web Services standard, or IIOP [17] for the Common Object Request Broker Architecture (CORBA), are supported. In [4] different approaches are discussed, which pursue the objective of establishing the CORBA standard also in the area of mobile applications.

The current version of our framework per default uses the SOAP protocol. CORBA functionality was realized by means of an own Object Request Broker, which is just going through its proof testing phase. Furthermore, the API of the communication framework permits the integration of additional communication protocols. This way, also future developments can be taken into account. A similar objective is also pursued by the OpenORB project [13], in which not the development of a complete architecture is in the foreground but only the middleware aspect is examined.

5. Conclusions

In this paper, we have introduced a generic architecture for the development of context-sensitive mobile applications. In addition to the flexible processing of context information, tracing back to the different ways of context recording, we also have described different techniques, which aim at conserving the resources and lowering the energy consumption of mobile terminals.

The ability to download components when required represents another option, which contributes on the one hand to conserve the limited storage of a mobile terminal and which, on the other hand, permits to adapt the application to the current user context. In addition, this technique has advantages in case that the connection of the mobile application breaks down. If the just required component should already have been installed on the mobile terminal, then it can be employed by the user also in the case of an interruption of the communication.

The described GUI framework overcomes the heterogeneity of the different Java libraries and takes into account the different technical conditions of mobile terminals. The presentation of results takes place according to the user settings. In addition to this, the framework offers functionality, which permits automatic adaptation of the user interface in case of context changes.

The generic communication framework supports different communication protocols at the same time and thus lays the basis for the integration of mobile context-sensitive applications with existing information systems.

With the variety of the research problems simultaneously tackled, the architecture introduced here offers a sound basis for the development of mobile context-sensitive applications.

Acknowledgements

This work was funded in part by Deutsche Forschungsgemeinschaft.

References

[1] K. Arabshian and H. Schulzrinne, GloServ: Global Service Discovery Architecture, *First Annual International Conference on Mobile and Ubiquitous Systems: Networking and Services* (*Mobiquitous*), Boston, USA, 2004.

[2] M. Aleksy, C. Atkinson, P. Bostan, T. Butter and M. Schader, Interaction Styles for Service Discovery in Mobile Business Applications, in *Proceedings of the 17th International Workshop on Database and Expert Systems Applications* (*DEXA 2006*) / *9th Workshop Network-Based Information Systems* (*NBiS'06*), 4–8 September 2006, Krakow, Poland, IEEE Computer Society, 60–65.

[3] M. Aleksy and M. Schader, Patterns for Mobile Applications, in: *Encyclopedia of Mobile Computing & Commerce,* Vol. 1, Idea Group, 2007, pp. 744–748.

[4] M. Aleksy, A. Korthaus and M. Schader, CORBA on Mobile Devices, in: *Encyclopedia of Mobile Computing & Commerce,* Vol. 2, Idea Group, 2007, 160–164.

[5] H.H. Bauer, T. Reichardt and A. Schüle, User requirements for location based services, in: *Proceedings of the IADIS International Conference E-Commerce 2005,* Vol. 2, 15–17 December, Porto, Portugal, IADIS, 2005.

[6] H.H. Bauer, T. Reichardt and A. Schüle, Was will der mobile Nutzer – Forschungsergebnisse zu den Anforderungen von Nutzern an kontextsensitive Dienste, in: *Aktuelle Trends in der Softwareforschung,* Heidelberg, 2006, 179–191.

[7] B. Bos, H.W. Lie, C. Lilley and I. Jacobs, Cascading Style Sheets, Level 2; CSS2 Specification. W3C Recommendation, World Wide Web Consortium (W3C), 1998.

[8] T. Butter, M. Aleksy, P. Bostan and M. Schader, Context-aware User Interface Framework for Mobile Applications, in *Proceedings of the 27th International Conference on Distributed Computing Systems Workshops* (*ICDCS Workshops 2007*) / *9th International Workshop on Multimedia Network Systems and Applications* (*MNSA-2007*), 25–29 June, 2007, Toronto, Canada, IEEE Computer Society.

[9] T. Butter, I. Duda, M. Aleksy and M. Schader, A Framework for Context-Sensitive Mobile Applications, in: *Proceedings of the IADIS International Conference E-Commerce 2006,* 9-11 December, Barcelona, Spain, IADIS, 308–312.

[10] T. Butter, S. Deibert and F. Rothlauf, Using Private and Public Context – An Approach for Mobile Discovery and Search Services, MMS 2006, in: *Mobile Informationssysteme - Potentiale, Hindernisse, Einsatz, 1. Fachtagung Mobilität und Mobile Informationssysteme* (*MMS*), 20–22 Februar 2006, Passau, Germany. LNI 76 GI 2006, 144–155.

[11] A.K. Dey, Understanding and Using Context, *Personal Ubiquitous Computing* **5**(1) (2001), 4–7.

[12] E. Gamma, R. Helm, R. Johnson and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software,* Addison-Wesley Longman, 1995.

[13] P. Grace, G. Blair and S. Samuel, A Marriage of Web Services and Reflective Middleware to Solve the Problem of Mobile Client Interoperability, in *1st International Symposium on Information and Communication Technologies,* Dublin, Ireland, 2005.

[14] M. Gudgin, M. Hadley, N. Mendelsohn, J.J. Moreau, H.F. Nielsen, A. Karmarkar and Y. Lafon, W3C SOAP Version 1.2 Part 1: Messaging Framework (Second Edition), W3C Recommendation, 27. April 2007, http://www.w3.org/TR/soap12-part1/.

[15] M. Kircher and P. Jain, Lookup, in: *Proceedings of 5th European Conference on Pattern Languages of Programs* (*EuroPLoP 2000*), 5–9 July 2000, Irsee, Germany.

[16] NAICS – North American Industry Classification Standard, http://www.census.gov/epcd/www/naics.html.
[17] Object Management Group, The Common Object Request Broker: Architecture and Specification. Version 3.0.3. OMG Technical Document Number formal/04-03-01, 2004, ftp://ftp.omg.org/pub/ docs/formal/04-03-01.pdf.
[18] A.W. Scheer et al., Das mobile Unternehmen, in: *Mobile Commerce – Grundlagen,* Geschäftsmodelle, Erfolgsfaktoren, Wiesbaden 2002, 91–110.
[19] Sun Microsystems Inc. J2ME Connected Limited Device Configuration. 2004.
[20] V. Venkatesh, V. Ramesh and A.P. Massey, Understanding usability in mobile commerce, *Communications of the ACM* **46**(12) (2003), 53–56.
[21] XUL Tutorial, 2006, http://www.xulplanet.com/tutorials/xultu/.

Markus Aleksy studied Management Information Systems at the University of Mannheim, Germany and holds doctorate degrees from the University of Mannheim and from the Tokyo Denki University. His research interests include Analysis, Design, Implementation, and Evaluation of Mobile and Distributed Systems.

Thomas Butter studied Management Information Systems at the University of Mannheim, Germany and currently works towards a doctorate degree at the University of Mannheim. His research interests include Software Design and Mobile Systems.

Martin Schader studied Industrial Engineering, Operations Research, and Information Systems at the Technical University of Karlsruhe, Germany. Currently, he is Chair of Business Information Systems at the University of Mannheim. His research interests include Analysis, Design, Implementation, and Evaluation of Mobile and Distributed Systems.