# AN ADAPTIVE MIDDLEWARE FOR MOBILE INFORMATION SYSTEMS

Volker Gruhn and Malte Hülder
*Applied Telematics/e-Business Group, Dept. of Computer Science, University of Leipzig, Klostergasse 3, Leipzig, Germany*
*gruhn@ebus.informatik.uni-leipzig.de,huelder@ebus.informatik.uni-leipzig.de*

Keywords: Mobile applications, Dynamic reconfiguration, Reconfiguration triggers, Adaptive middleware, Reactive systems.

Abstract: The advances in mobile telecommunication networks as well as in mobile device technology have stimulated the development of a wide range of mobile applications. While it is sensible to install at least some components of applications on mobile devices to gain independence of rather unreliable mobile network connections, it is difficult to decide about the suitable application components and the amount of data to be provided.Because the environment of a mobile device can change and mobile business processes evolve over time, the mobile system should adapt to these changes dynamically to ensure productivity.
In this paper, we present a mobile middleware that targets typical problems of mobile applications and dynamically adapts to context changes at runtime by utilizing reconfiguration triggers.

## 1 INTRODUCTION

Over the past years, the development of mobile applications has increased significantly, and more and more specialized applications are introduced that support mobile business processes performed by mobile field workers.

For example, let us consider an information system that enables an insurance agent to perform his business processes. Supplementing office work, the agent visits his clients at their homes or at the site of insured objects. Particularly in claims management, he has to visit the insured object, ascertain necessary data and organize the claim settlement. Depending on the type of claim, the claims management process can be very different: for a damaged car, e.g., in a catalog containing all parts of the particular model, the affected parts have to be marked. For other objects, the agent has to take photos of the affected objects.

From this example, we can see two major issues that need to be considered in supporting mobile business processes (Gruhn et al., 2007): On one hand, the processes may be very specific and complex, and on the other hand, data used during a process may only be used for this particular process (like the catalog containing all parts of a particular car). Traditionally, these issues have been addressed in two ways:
1. A complex application is installed on the mobile device, which comprises the knowledge of all processes and their necessary data. This approach allows working with the application autonomously, but it may also consume a lot of storage space on the mobile device. Moreover, it is rather expensive in development, operation, and maintenance.
2. Alternatively, the complex application is installed on a server, while the mobile device only holds a thin client that only renders the data. Resource consumption on the mobile device, as well as development and maintenance costs, remain low. However, such a thin client application depends on a permanent network connection to the server (Jing et al., 1999).

Our approach proposes to dynamically mobilize components and data at runtime, based on knowledge of business processes and the current environment context. We therefore present a mobile, service-oriented middleware that allows transferring applications (or parts of them) from a server to mobile client devices. The middleware provides an execution environment that allows execution of the same implementation on both the server and the mobile clients. It also dynamically adapts to the current environment context, e.g. it considers the current network connection when invoking services on the local device or the remote server. The considered context information, possible adaptations, and the rules controlling dynamic reconfiguration are presented in sect. 4.2.

In sect. 2, we discuss typical problems experienced by mobile applications, followed by an outline of related work (sect. 3). Subsequently, we introduce our mobile middleware in sect. 4, which is completed by a validation in sect. 5. The paper is rounded off by our conclusions and future outlook (sect. 6).

## 2 PROBLEMS OF DISTRIBUTED MOBILE APPLICATIONS

One of the most crucial problems of distributed mobile applications is volatile network connectivity. Mobile network connections are rather slow compared to wired networks and also they often vanish without warning. Applications depending on a network connection cannot be used under these circumstances.

On the other hand, mobile information systems may be very helpful, if they can be used in places where no network connection is available, e.g. in a building's basement, out in the field, or even in a disaster area. Therefore, mobile information systems should be capable of disconnected operation.

Another crucial problem is the limited storage space of mobile devices. The limited resources do not allow placing a complex application and all of its data on a mobile device. Hence, only selected parts of complex applications can be made available on the device to support the current mobile business process. Similarly, only a carefully chosen subset of the data in a central data store can be replicated to a mobile device to support these activities.

In both cases, it is difficult to decide about the appropriate selection of data and components which ought to be transferred to the mobile device. Furthermore, such a decision should not be fixed, but has to be made repeatedly to adapt to the current situation of the device and the mobile business process.

Another challenge lies in the fact that mobility and size of mobile devices causes them to be lost or stolen more easily than fixed office computers. Providing applications and data on a mobile device brings about increased security requirements. For example, confidential data should be removed from the device when it gets lost or at least the data has to be encrypted so that it is useless for an unauthorized finder.

Applications installed on mobile devices may have to be updated from time to time. Calling the devices in to be updated by an administrator is time-consuming and requires logistical effort. Distributing updates on compact discs or other media requires the users to be able to perform the updates themselves and increases the risk of updates being installed with significant delays or not at all. Thus, online updates via the Internet have become popular. However, providing an update management tool for each mobile application individually, increases development effort and resource consumption on mobile devices.

Advances in mobile device technology have brought about numerous different devices and an increasingly faster development of even more device types. Keeping up with this development by providing applications tailored for specific devices becomes more and more complex and expensive. An execution environment that levels device differences may reduce development and maintenance costs significantly.

We consider these problems relevant for such a number of mobile applications that they ought to be addressed in a general way. Our approach is based on an adaptive mobile middleware, which targets the stated problems for the class of mobile information systems, and is variable enough to support a great variety of mobile business processes. Sect. 4 describes the main aspects of our approach in more detail.

## 3 RELATED WORK

In this section, we give a brief overview on selected related work:

Hong et al. give an introduction to context-awareness. They group context information in computing, user and physical context (Hong et al., 2005). We do not see the necessity to handle these three types of context differently, and therefore we propose a generic context interface for components and reconfiguration triggers. While currently only the mobile business process context and the most important computing context sensors (CPU, memory, disk space, bandwidth) have been implemented, other sensors may easily be added at a later stage.

Coda (Kistler and Satyanarayanan, 1992) is a distributed file system that provides file operations for mobile systems which are temporarily disconnected, by implementing client-side caching, and server replication mechanisms. Files can be stored on several servers so that a client may copy them from another server, if one is not available. However, Coda is based on the notion of files, which is too coarse-grained to support service orientation and synchronization of business object attributes.

The Odyssey system (Noble and Satyanarayanan, 1999) has evolved from Coda. While Coda hides environment data from applications, Odyssey allows quality-based access to data, notifying the application when a parameter (e.g. the available network bandwidth) is out of bounds and thus enabling it to react to this situation accordingly. However, Odyssey

is also based on the notion of binary files which is not suitable for service orientation and synchronization of fine-grained business object attributes.

Other than Coda and Odyssey, Xerox PARC's Bayou (Terry et al., 1998) does not try to resolve data inconsistency itself. It rather notifies the applications which themselves can provide methods to resolve the conflicts. Like other file-based systems, Bayou has not been designed for service-oriented architectures.

The SATIN component system (Zachariadis et al., 2006) follows a similar approach to ours of providing a light-weight middleware for mobilizing application parts. However, we put a strong emphasis on service orientation as well as managing user data during offline operation. Furthermore, we propose a solution to decide about mobilizing components at runtime, based on knowledge about mobile business processes.

Popa et al. introduce a mechanism called "code collection" for code execution on mobile devices in (Popa et al., 2004). Code collection is a management concept combining caching techniques and garbage collection techniques for memory management. It dynamically loads and removes methods to and from the device's memory depending on the amount of memory available. Similar to data caching, subsequent calls to removed methods lead to a miss and require reloading those methods via the network. Thus, it is not suitable for offline operation. We still consider this approach interesting as it may help to improve the selection of components most likely being used during disconnected mode as we discuss in sect. 6.

A rather recent development is Google Gears (Google, 2007; Herrington, 2007). It solely targets web applications for which it supports an offline mode. The execution of business logic in offline mode is restricted to logic implemented in JavaScript which can be executed in the browser. In our approach we focus on mobilizing and executing the same implementations on server and mobile clients.

# 4 MobCo MIDDLEWARE

Our mobile middleware was developed as part of the *MobCo* project. It is implemented on top of the OSGi framework standard (OSGi Alliance, 2005), which itself is a service oriented component layer based on the Java programming language.

## 4.1 Subsystems

The architecture of *MobCo* middleware is shown in Figure 1. It comprises a number of subsystems, each of which is responsible for different aspects.

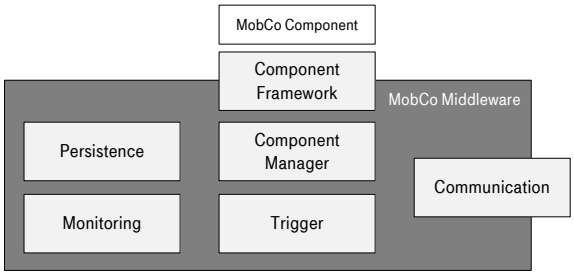


Figure 1: Subsystems of *MobCo* Middleware

The **component framework** provides the interface used by components to interact with the mobile middleware. For example, it allows a component to lookup and invoke services provided by other components. It also provides access to the persistence subsystem allowing a component to query, load and store data, and to the monitoring system allowing to inspect the current environment's context.

The **component manager** is responsible to deployment and removal of components. On that basis, it also has to manage the transfer of new or updated components and control the update process of undeploying an outdated and deploying the updated version of a component. The component manager also resolves component dependencies and automatically transfers required dependencies to the mobile system.

The **persistence** subsystem realizes a distributed data store which consists of a local data store on each mobile device and a central database on the server. It allows selecting a number of data objects to be replicated to the mobile device and synchronized with the server when necessary.

The persistence subsystem uses the object oriented database db4o (Versant Corp., 2008) as a basis for its functionality, but it hides the db4o interfaces behind the component framework. This way, the database may be substituted, if necessary.

The **communication** subsystem controls the communication between two middleware instances and it controls the routing of service calls by components.

The R-OSGi extension (Rellermeyer et al., 2007) is used for the incorporation of remote services provided on another device.

The **monitoring** subsystem collects information about the system's environment. In particular, it observes the device's network connectivity, its storage space, memory and computational load.

The **trigger** subsystem manages the dynamic reconfiguration of our mobile middleware by means of reconfiguration triggers, which decide about necessary reconfigurations based on the information collected by the monitoring subsystem.

## 4.2 Dynamic Reconfiguration

Dynamic reconfiguration means changing the distribution or the behavior of a mobile application at runtime, according to changes in the environment. As we have seen in sect. 2, mobile applications often suffer from similar problems and thus, they may adapt to environment changes in similar ways. Our middleware provides several standard reconfigurations, some of which can be applied independently of the current application, and some that may be utilized by application developers for their individual applications.

In order to design the ways in which such a dynamic reconfiguration takes place, we have to define the context information to react to, and the set of possible reactions which can be performed according to the current situation. We call the rules that decide about whether a reaction has to follow the current situation *reconfiguration triggers* or just *triggers*.

Triggers collect information about the environment and the executed mobile business processes and decide whether a reconfiguration is necessary. Thus, triggers provide a mapping from the set of environmental and mobile business process information to the set of possible reconfigurations, which will be defined in the following sections.

### 4.2.1 Relevant Context Information

For the sake of brevity, we will only name (rather than discuss) the parameters that are considered for the set of mobile business processes and environmental information that triggers base their decisions on: Network type, bandwidth, latency, CPU power, CPU load, device type, storage space, memory, component size, component dependencies, component version, user information, and the currently executed mobile business process.

### 4.2.2 Possible Reconfigurations

The set of reactions that may be initiated by reconfiguration triggers can be divided into reactions that have to be provided by a mobile middleware, and reactions that change the behavior of an application. In order to keep the middleware small and lightweight for mobile devices, the set of middleware reactions contains only the reactions necessary for dynamic changes in component distribution, dynamic routing of service calls and handling of replicated data. This comprises the following activities:

**Mobilization.** A trigger may decide to mobilize a component, i.e. transfer it to the mobile host for local execution.

**Removal.** A trigger can react to outdated or unnecessary components or low storage space by removing a component from the local system.

**Routing.** Components can reside on a mobile host as well as on the server. A trigger may decide for the application to employ the local or the remote installation, depending on the current situation.

**Replication.** The trigger can initiate the replication of selected business objects, when a connection becomes available, or when a replicated business object is changed on the server.

**Synchronization.** When a business object has been changed during offline mode and a network connection becomes available, synchronization of the replicated business object has to be initiated. During synchronization, possible conflicts and inconsistencies between local and remote copy have to be resolved.

**Application Reaction.** If a mobile application provides its own adaptations to different situations, triggers can be used to invoke the according functionality in the appropriate situation.

## 4.3 Standard Triggers

Based on the monitored environment data and the possible reconfigurations, we have implemented a number of standard triggers, which can be configured for individual application needs, if required.

Application developers may implement their own reconfiguration triggers to control dynamic adaptation of the middleware, or to allow their applications to adapt to context changes by application reactions.

**Network Class Triggers.** are used to group low level network information like network type, latency and bandwidth into high level network classes. Network classes may be defined by the developer, e.g. ”low speed”, ”high speed”, or ”WLAN”, ”GPRS”, and ”UMTS” could be such classes.

**Routing Triggers.** adjust the preferred routing of service calls based on the current network class, the availability or unavailability of certain components, or the occurrence of certain process events.

**Offline Readiness Triggers.** control the mobilization of components to ensure their availability during offline mode. There are Offline Readiness Triggers that mobilize components necessary for complete applications, selected mobile business processes, or according to a developers’ selection.

**Update Triggers.** control checking and provision of updates. If updates are available, selective triggers

can prioritize them according to their relevance for the currently executed application or process.

**Download Management Triggers.** control enqueuing component downloads to the download queue, aborting currently enqueued downloads, and adjusting the number of parallel downloads.

# 5 EXPERIENCE

To show the feasibility of our approach, we have implemented the mobile middleware as described above, and we have built a mobile information system application that utilizes the middleware's features.

## 5.1 About the Application

The application is taken from the telecommunications domain and supports service technicians performing their work in the field. In particular, service technicians have to visit customers and install or repair their telephone or Internet connections, and they have to maintain infrastructure appliances.

The application comprises more than 90 components that provide small application units to allow for fine-grained distribution on mobile devices.

Currently, computers running Microsoft Windows XP and Vista, MacOS 10.5 and mobile devices running MS Windows CE and Mobile are supported.

## 5.2 Mobile Business Process

The mobile business processes supported by the application are easy to understand. Still, the complete mobile business process model which provides the foundation for our implementation is too complex and beyond the scope of this paper.

In brief, the service technician receives a number of tasks from his dispatcher on his mobile device. Before he starts his work, the service technician sorts the tasks in a convenient order he intends to follow for the rest of the day. This sequence is reported back to the dispatcher who may assign additional tasks during the course of the day, if they fit the technician's plan, e.g. because they are close to the current task's location.

During his working day, the technician attends the customers' locations and installs or repairs their telecommunication connections. On completion of such a task, he files an activity report and updates the vehicle log. Additionally, he may give advice on additional equipment (e.g. a WLAN router) and order appropriate devices from an online procurement system. As the service technician has direct customer contact, he may also perform a customer survey.

## 5.3 Component and Data Distribution

From the mobile business process model, the mobile middleware gains knowledge about the components necessary to execute a certain process. This knowledge is used to provide the relevant components on the local device in time, when the process is started. This way, it can be assured that all necessary components are locally available to execute and complete the current process, even if the device gets disconnected.

## 5.4 Update Management

The application uses an update trigger that checks for updates upon starting the middleware, typically on a daily basis. The service technician performs the customer survey after the execution of a task that involves customer contact. Therefore, when such a task is executed, another update trigger checks for updates and ensures that the latest version of survey components is provided before the survey is to be performed.

## 5.5 Utilizing Context Information

The developer only has to care about component locations and routing of service calls, if the application shall behave in different ways depending on the environment's context. In our example, the application comprises a simple procurement system that allows ordering additional telephone or computer equipment.

If the device has an online connection to the sever when the procurement system is accessed, the list of products can be loaded from the server including up-to-date price and stock information. If the device is disconnected when the procurement system is accessed, only the ten most popular products can be loaded from the local data store. The number of products stored on the mobile device has been reduced significantly to save local storage resources, while still providing basic procurement system functionality in disconnected mode. Additionally, there is no up-to-date information about the availability of products in stock. Therefore, the procurement system has to behave differently depending on the network connection: In online mode, the order can be placed directly on customer's request, but in offline mode, only a quotation for the selected products can be requested. This request is stored locally on the device until it reconnects to the server. After reconnection, the quotation request is transmitted to the server which processes the request and creates an individual quotation including the current price list and stock information.

Note that this use of context information is application-specific – customers' satisfaction is ex-

pected to increase when an order can only be placed if up-to-date pricing and stock information is available.

As we have seen, our middleware cares about incorporating local or remote services automatically, based on the current context and developers' preferences. But it allows applications to observe the current context and adapt their behavior as well.

# 6 CONCLUSIONS AND OUTLOOK

We have developed a middleware for mobile devices that allows execution of application components on mobile hosts as well as dynamic reconfiguration of its behavior by evaluating triggers and realizing their reactions. On top of that, we have implemented a mobile application from the telecommunications domain utilizing the mobile middleware's features.

The results show the feasibility of our approach. Instead of solving the typical problems of mobile applications named in sect. 2 from scratch, they have been targeted by our mobile middleware. Application developers therefore may concentrate on the actual business logic of their mobile information systems.

While the validation has shown the approach to be feasible, both mobile middleware and the implemented application have not yet reached production status. Before the middleware may be used in productive environments, we have to increase its stability and user friendliness, and we have to address some security and performance issues.

For future research, we see a great potential in actively utilizing mobile business process models to transfer the required components to the mobile device early during process execution, to allow for seamless operation in offline mode. Currently, the selection of needed components has to be done manually by implementing an appropriate trigger, or automatically based on dependencies.

Popa et al. propose the inspection of code dependencies and object references to reclaim memory (like in garbage collection) to be combined with data about invocation frequencies of methods (similar to caching strategies) (Popa et al., 2004). We believe that enriching code dependencies, object references, and mobile business process information with user behavior information may additionally aid the decision about mobilizing the code that is likely to be used in future. Proving this hypothesis is subject of future research.

# ACKNOWLEDGEMENTS

The Applied Telematics/e-Business group is endowed by Deutsche Telekom AG. The mobile middleware was developed in the *MobCo* project which was cofounded by Deutsche Telekom Laboratories.

# REFERENCES

Google (2007). Google Gears. http://gears.google.com/.

Gruhn, V., Köhler, A., and Klawes, R. (2007). Modeling and analysis of mobile business processes. *Journal of Enterprise Information Management*, 20(6):657–676.

Herrington, J. (2007). The Power of Google Gears. http://www.oreilly.de/artikel/2007/07/googlegears.html.

Hong, D., Chiu, D. K. W., and Shen, V. Y. (2005). Requirements elicitation for the design of context-aware applications in a ubiquitous environment. In *Proceedings of the 7th international conference on Electronic commerce (ICEC '05)*, pages 590–596, NY, USA.

Jing, J., Helal, A. S., and Elmagarmid, A. (1999). Client-server computing in mobile environments. *ACM Comput. Surv.*, 31(2):117–157.

Kistler, J. J. and Satyanarayanan, M. (1992). Disconnected Operation in the Coda File System. *ACM Transactions on Computer Systems*, 10(1):3–25.

Noble, B. and Satyanarayanan, M. (1999). Experience with adaptive mobile applications in Odyssey. *Mobile Networks and Applications*, 4.

OSGi Alliance (2005). OSGi Service Platform - Release 4. Technical report, OSGi Alliance.

Popa, L., Raiciu, C., Teodorescu, R., Athanasiu, I., and Pandey, R. (2004). Using code collection to support large applications on mobile devices. In *MobiCom '04: Proceedings of the 10th annual international conference on Mobile computing and networking*, pages 16–29, New York, NY, USA. ACM.

Rellermeyer, J. S., Alonso, G., and Roscoe, T. (2007). R-OSGi: Distributed Applications Through Software Modularization. In *Middleware '07*, volume 4834 of *LNCS*. Springer.

Terry, D. B., Petersen, K., Spreitzer, M. J., and Theimer, M. M. (1998). The Case for Non-transparent Replication: Examples from Bayou. *IEEE Data Engineering*, pages 12–20.

Versant Corp. (2008). db4objects. http://www.db4o.com/.

Zachariadis, S., Mascolo, C., and Emmerich, W. (2006). The SATIN Component System–A Metamodel for Engineering Adaptable Mobile Systems. *IEEE Transactions on Software Engineering*, 32(11):910–927.