HYBRID APPLICATION SUPPORT FOR MOBILE INFORMATION SYSTEMS

Volker Gruhn, Malte Hülder
Chair of Applied Telematics / e-Business, Dept. of Computer Science, University of Leipzig
Klostergasse 3, 04109 Leipzig, Germany

Keywords: Pervasive/ubiquitous computing, distributed systems, e-commerce/m-commerce, mobile code and systems, service continuity.

Abstract: The wide-spread presence of wireless networks and the availability of mobile devices has enabled the development of mobile applications that take us a step closer to accomplishing Weiser's vision of ubiquitous computing (Weiser, 1991). Unfortunately however, network connectivity is still not given anywhere and at any time. To increase the benefit of mobile applications, the next logical step is to provide support for an offline modethat allows to continuously work with an application, even when the device is not connected to a network. In this paper typical problems of replicating data are explained, possible solutions are discussed and two architectural patterns that could be used to implement hybrid support are illustrated.

1 INTRODUCTION

Recent years have brought a wide-spread presence of wireless networks and a diverse range of mobile devices, pushing forward the development of mobile applications significantly. For quite a number of applications it is feasible to install the application on the mobile device. For example, a very popular application used on personal digital assistants (PDAs) is personal information management, usually containing calendar, address book, and notepad applications. To manage the data, a number of users synchronize it between PC and PDA. For certain purposes, one synchronization architecture may be more suitable than another, as pointed out in (Starobinski et al., 2003).

Mobile devices have very limited resources. For some applications it is therefore more difficult to synchronize data needed for mobile use. For example, consider an information system that enables construction supervisors to carry electronic copies of project documents while visiting a construction site: Due to limited memory of the mobile device, the supervisors have to carefully select which documents to download, and for another construction site, the downloaded documents have to be deleted in order to free up memory for other documents.

Thus, it appears to be very sensible to cope with these limitations by placing functionality and application logic on a server. On the mobile device resides a thin client (Orfali et al., 1996), often a browser, which exclusively renders data. Instead of selecting and downloading data before he attends the place of work and risking to make a wrong or incomplete selection, the user can choose which data is needed just in time, and only this data is transferred. The application thus depends on a permanent network connection to the server. Even for traditional web applications for PCs with a wired connection, network problems like congestion are problematic. For mobile devices that can connect only by wireless means, there is also the problem of complete disconnection. If a disconnection is due to the limited coverage of mobile telecommunication networks, the problem cannot be solved quickly by dialing in again.

To keep working with the application in such a case, the application should be be able to switch into an offline mode. While in offline mode, the necessary application data and application logic will have to be available in order to continue working.

For this paper, we want to focus on mobile information systems with distinct client-server characteristics since a large base of such information systems, albeit without mobile capabilities, already exists today. Consequently, the implications of enabling mobile access to existing information systems are an important issue in the industry.

Gruhn V. and Hülder M. (2005).
HYBRID APPLICATION SUPPORT FOR MOBILE INFORMATION SYSTEMS.
In *Proceedings of the Seventh International Conference on Enterprise Information Systems*, pages 232-237
DOI: 10.5220/0002522802320237

2 RELATED WORK

The Rover Toolkit (Joseph et al., 1995) describes the possibilities that become available by the use of *queued remote procedure calls* (QRPCs) and *relocatable dynamic objects* (RDOs). These mechanisms work for simple applications like e-mail and web browsers, where calls can be queued in offline mode according to the store-and-forward principle and executed after switching to online mode. When the paper was published in 1995, users typically requested statical web pages which were delivered by the web server. However, complex applications are available today that can be entirely controlled in a web browser. For every single interaction step, the web server generates a new page, which cannot be stored in cache because entering different data will lead to different pages and thus to cache misses. The main ideas of this work are still relevant, but have to be accomplished by other means.

(Weinberg and Ben-Shaul, 2002) show a number of mechanisms which extend the FarGo-framework (Holder, 1998) to FarGo-DA. Although some points are discussed that will also be picked up in this paper (especially in section 4.2), it is always assumed that either application or framework know when a change between online and offline mode and vice-versa is about to happen, so that necessary synchronizations can take place in time. These prerequisites are not given in a mobile environment, because it cannot be foreseen when a wireless connection goes down. Therefore, a framework for mobile applications should support an offline mode even when disconnections are unforeseeable.

With μCODE, a Java framework based on the Java Aglets API is proposed in (Picco, 1998). It includes a "MuServer" and several interfaces that have to be implemented in order produce mobile code. The author assumes that it is sensible to move certain classes to the client and execute them there in order to reduce communication costs. This idea has to be extended to the possibility of an offline mode, which allows for working in a disconnected environment.

Jørstad et al. describe service continuity for generic mobile services in (Jørstad et al., 2004a) and (Jørstad et al., 2004b). However, due to their focus on generic mobile services, it does not become clear whether their ideas are particularly suitable for mobile information systems. We will consider some of their ideas in more detail in section 5.

3 TERMS AND DEFINITIONS

In this section, we want to introduce some terms and definitions used in the rest of this paper.

3.1 Online, Offline, Hybrid

(Merriam-Webster, 2003) defines *online* as "connected to, served by, or available through a system and especially a computer or telecommunications system (as the Internet)". Accordingly *offline* is defined as the opposite: "*not* connected to or served by a system and especially a computer or telecommunications system". An *online application* is therefore an application that offers its services only when connected to such a telecommunications system, whereas an *offline application* does not make any use of such a connection. A *hybrid application* is an application that combines features of offline and online applications: it offers some services only when it is connected, while other services are available even when there is no such connection. Let us consider an e-mail application as an example: when the application is in *online mode* (i.e. connected to a telecommunications system), it can send and receive new e-mails. When the application is in *offline mode* (i.e. disconnected), the user may still read messages that were downloaded before entering offline mode, and he may still write messages that will be saved locally until entering online mode. A hybrid application therefore needs to be aware of its connection state. Switching from online mode to offline mode can be done manually, i.e. initiated by the user, or automatically by some connection detection algorithm. One advantage of manually switching from online mode to offline mode is the possibility to perform certain tasks before actually entering offline mode, e.g. downloading the latest e-mails. Another one is the possibility to disconnect deliberately in order to save online costs and battery power. A disadvantage is the necessity of user interaction – if the user does not switch to offline mode before the connection breaks down, the application might malfunction as it may assume a working connection. Therefore, developers of hybrid applications should aim to provide an automatic connection detection and also allow for manual mode switching where possible.

3.2 Different Kinds of Mobility

When thinking of mobile applications, the following definition (from (Merriam-Webster, 2003)) of *mobile* as "capable of moving or being moved" springs to mind. But which entity exactly is capable of moving or being moved in context of mobile applications? It could be the user, it could be the device, and it could also be the program code of the application. In this context, (Pandya, 2000) discerns personal mobility, terminal mobility, and service portability: According to his definition, personal mobility means that the user is not restricted to a single device, but can rather use different devices in order to use a service. Thus it does not mean the user's capability of moving physically, but the possibility to "move" between different devices. Terminal mobility is given when the device (the terminal) remains connected to a network while it is moved physically. Finally, a user experiences service portability when he may access the same service from anywhere, independently of his location or the device he is using. (Roth, 2002) gives the example of e-mail access from any location in the world for service portability.

The notion of service portability is a bit tricky, as it depends on the service, in which way service portability can be achieved. Considering the example of e-mail access, we can see the following: As long as the user stays in an area where there are enough devices to let him access his e-mails at any time, personal mobility could suffice to achieve service portability as well. As soon as the user wants to access his e-mails beyond that area or in between the locations where suitable devices are installed, terminal mobility is needed. The user then carries the device to any place where he wants to access his e-mails. But true terminal mobility is still not possible: On air, in forests, or even underground, device connectivity is not available. In many cases, true terminal mobility is not necessary as long as the device behaves in a feasible way while it is disconnected.

It should have become clear that true service portability cannot be achieved, since it is impossible to provide connected devices in every location, and also since true terminal mobility cannot be guaranteed. Therefore, a way should be found to provide as many services as possible, even when disconnected.

3.3 Mobile Code

A possible solution to overcome the problem of limited terminal mobility could be *mobile code*, which is defined by (Adl-Tabatabai et al., 1996) as "any program that can be shipped unchanged to a heterogeneous collection of processors and executed with identical semantics on each processor." In times when device connectivity is not available, mobile code might be shipped to the device and executed there in order to provide the service that would otherwise not be available in offline mode. When talking about mobile code, two other terms are used frequently:

A *mobile agent* is a unit of modularity, execution and mobility that can migrate from one mobile host to another (Julien and Roman, 2002). A *mobile host* is a container for mobile agents characterized, among other things, by its location (Julien and Roman, 2002). Put in other words: A mobile host is a device that can be moved physically and which provides an environment where mobile agents can be executed.

This definition by (Julien and Roman, 2002) does not include the need for mobile agents to autonomously migrate from one mobile host to another. In the context of mobile information systems, this does not pose any problems. In fact, a user is interested in using a service continuously with his device, even when there is no network connection available. Therefore, the necessary mobile code should have been moved to the mobile host before the connection was lost – which part of the application decided to move this component is not of interest to him. The notion of a mobile agent in this context is thus interesting from the component's point of view. A mobile agent is a unit that combines all the classes or components needed to provide a service, possibly also in combination with the data needed. The problems that arise from copying mobile agents and their data to mobile hosts will be explained in the following section.

4 TYPICAL PROBLEMS WITH REPLICATED DATA

When working with local copies of data (replicas), there are some typical problems. One of them is the possibility that data is not available when it is needed: Especially due to limited storage space or network bandwidth, not all data can be downloaded to the client, and thus only a restricted set of data can be accessed during offline mode. Therefore, it is necessary to reliably predict which data will be needed in offline mode, and to cope with situations when needed data is not available, e.g. by disabling a service. If offline mode was activated manually and not caused by network problems, the application could also suggest to switch to online mode and access the needed data via the network (if still available).

4.1 Deciding on Data Transfer

Predictions which data is needed during offline mode can be made manually, e.g. the user chooses the priority for services and data he wishes to have available in offline mode. The software will then try to transfer data with highest priority first. Data with lower priority will be transferred only when there is enough time, bandwidth and storage space left. Alternatively, it might be possible for the software to decide automatically which data a user will use most likely by monitoring user interaction over some period of time. Using such monitoring, one could not only identify the services used most frequently, but also the use in certain situations and even the data volume needed to transfer code and data. With this knowledge it should be possible to calculate a weighed probability (p) for each service to be used within a certain amount of time. From the typical data volume known for each

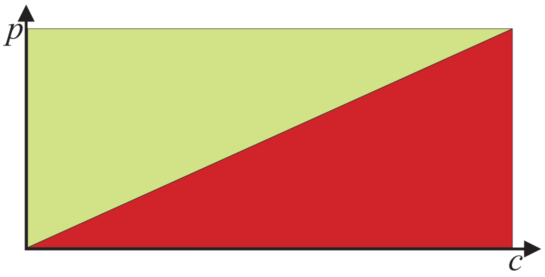

Figure 1: Function $f = c - pc < t$

service and from the current situation, one could calculate the cost (c) it takes to transfer code and data for a certain service. Only if the result of the function $f = c - pc$ is lower than a certain threshold value (t), code and data for this service are loaded (fig. 1). Thus it is assured that for a service that will probably be used $(p \approx 1)$, code and data are loaded $(f \approx 0)$, whereas a service that is less likely to be used $(p \rightarrow 0)$ will only be loaded, if the costs are low as well $(c \rightarrow 0)$.

4.2 Inner States

Because different instances of software are characterized by the possibility to incorporate different inner states, it has to be considered how to deal with these inner states. Problems can arise, if data is changed on the server while the client is in offline mode:

As long as this data is not the basis of other processes, there should be no problem – as soon as the application switches to online mode, it will use the new data available on the server. Next time it switches to offline mode, the current data will be downloaded.

If the changed data was used in other processes (e.g. certain calculations), those processes might have to be re-executed after switching to online mode in order to get the up-to-date results depending on up-to-date data. It will be an application-specific decision for each process, whether this re-execution is necessary or not. Sometimes the execution time of the process will be decisive, as the results were valid at the time they were processed, and the data was changed later – an execution in online mode at the same time would have given the same result.

Other problems may arise, if an inner state is changed during offline mode, as a change of inner state may be considered as a change of replicated data. When switching back to online mode, local changes have to be transmitted to the server. According to (Weinberg and Ben-Shaul, 2002), four ways to handle this problem can be described:

Purge means that changes of state are discarded. The client may call methods of the component and read its state but should not change it, as these changes are lost when switching to online mode.

Overwrite transfers the state of the client component to the server component. If several clients had copied the component locally, there might be inconsistencies when switching to online mode.

Merge is supposed to try to solve conflicts emerging from state changes. Unfortunately, there is no general strategy for coping with those conflicts, but the developer has to implement solutions when developing the component.

Last demands timestamps to be given at any change of state, so that after switching to online mode, the state with the youngest timestamp is copied to the server. Concurrent access may lead to inconsistencies as well, and the clocks on the server and all clients need to be synchronized.

Variants of the last alternative are also possible, e.g. the state with the oldest timestamp could be copied to the server, and thus the first change "wins". To show other possibilities, we want to distinguish two different cases:

1. **Data is changed on *one* client while it is in offline mode:** Assuming changing the data was an allowed action for the client, the data has to be transferred to the server after switching to online mode again. For other clients that used this data, this case is similar to the one where data was changed on the server. Some processes on the other clients may have to be re-executed. Another severe problem arises from this: If the changes a client has produced in offline mode can lead to the re-execution of processes performed by other clients in offline mode, what happens to processes performed by clients in online mode? Consequently, these processes would have to be re-executed as well, if the timestamp of the changed data is older than the execution of the process.
2. **Data is changed on *several* clients while they are in offline mode:** Different strategies can be applied to decide which changes have to be accepted by the server and the other clients. According to the solution with only one client, it can be checked when the change took place on the different clients. All processes executed before the first change will not be affected. Processes executed during the first and second change will have to be re-executed using data after the first change. Processes between the second and third change will have to be re-executed using data after the second change and so on. But the problem does not only lie in the number of changes, but also in the uncertainty of when a client switches back to online mode and requires synchronization. As long as a client that switched

to offline mode before another client remains in offline mode while the other client has switched back to online mode, any processes depending on data that the first client might have changed are still subject to possible re-execution later on. For a number of applications, this is not acceptable.

Another possible strategy could be to only allow the first client which synchronizes to deliver changes to the server. Other clients that also submit changed data to the server later on will be told that their changes are invalid and processes depending on those changes will have to be re-executed. The main advantage is that after switching to online mode and connecting to the server, it is known which processes have to be re-executed and that processes are to be re-executed at most once.

It should have become clear that multiple changes on data needed by several clients pose severe problems. If changes brought about by a client in offline mode demand re-execution of processes, the results of any process executed while another client is in offline mode may not be valid. This appears to be feasible for only very few applications, if any. It therefore seems to be sensible to restrict the services available in offline mode to those which cannot lead to the named problems. Also some optimistic strategy may be applied, as for many applications the probability of concurrent changes to the same data is very low. If such a conflict occurs, it has to be detected and changes by the different clients have to be discarded (except maybe for the first one).

5 ARCHITECTURAL PATTERNS

To support service availability during offline mode, two main architectural patterns spring to mind: a connection manager component or a service continuity layer as introduced by (Jørstad et al., 2004a). The connection manager component would be placed besides other components, which may or may not make use of the connection manager component. Thus, components offering one or more services need to actively make use of the connection manager component in order to provide their service in online mode or offline mode. In a layered approach, the idea is that any communication needs to go through the adjacent layer, so it should not be possible to bypass a layer. Therefore, components (or rather their developers) cannot decide whether to use the service continuity layer or not – as soon as a component needs to use a network connection, it will have to communicate with the service continuity layer.

A connection manager component would be placed besides other technical components. It can make use of those components, e.g. an "always best connected" component that detects available networks and determines which one is most suitable for a certain purpose in the actual situation. Other components, technical ones as well as those implementing application logic, may use the connection manager component to provide services in offline mode. This could be helpful when migrating an existing application because components could be migrated one after another, introducing offline mode capability where suitable, while other components will not (yet) be changed and thus will not (yet) be available during offline mode.

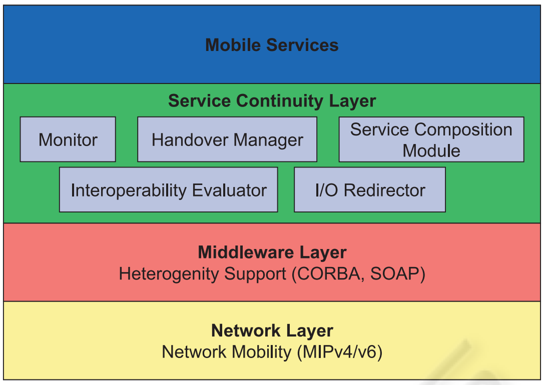
Mobile Services
Service Continuity Layer
Monitor
Handover Manager
Service Composition Module
Interoperability Evaluator
I/O Redirector
Middleware Layer
Heterogenity Support (CORBA, SOAP)
Network Layer
Network Mobility (MIPv4/v6)

Figure 2: Service Continuity Layer (Jørstad et al., 2004a)

The service continuity layer (fig. 2) proposed in (Jørstad et al., 2004a) and (Jørstad et al., 2004b) contains a monitor, a handover manager, a service composition module, an interoperability evaluator, and an I/O redirector. The monitor is to describe the surroundings of a host so that possible redistributions of a service can be identified, in order to provide service continuity. A handover manager can instruct the service composition module to design and implement a new service composition of an existing service as soon as it (or the monitor) recognizes that a handover will be necessary. The service composition module is supposed to compose a new service out of the components that are available. Available components are checked by the interoperability evaluator for whether they are compatible on both their interfaces and their behavior, so that it is safe to compose a service out of them. The I/O redirector redirects I/O between different service components, so that a service can be used continuously even when it is rearranged by the service composition module.

As already mentioned in section 3.1, we see a need for a monitor that observes network connectivity and initiates offline mode automatically when the connection is going to break down, or switches back to online modewhen a connection is available again. On the other hand, a handover manager should already be implemented in lower networking layers, e.g. when roaming between GSM cells is performed. A service composition module will only be necessary, when new services become available during runtime and have to be integrated with the services available already. This focuses on generic devices and generic services that might compose an application out of available services, like it is proposed for web services. For business applications, where a company equips their employees with devices and applications, this is usually not a requirement. The interoperability evaluator should check whether a new component can interact with the existing ones, and if it is safe to do so. We see a great overlap with the service composition module here. Again, this may be of interest in very heterogeneous environments – within a company, interoperability should already be ensured during development. The I/O redirector again seems to be a very important component, as it manages access to data sources and storage. It has to assure that data needed during offline mode is fetched from a local data storage, and that data changed during offline mode is stored in a local storage and transmitted to the server upon entering online mode.

6 CONCLUSIONS AND FUTURE WORK

As mobile hardware and communication costs are dropping constantly, many organizations may afford to equip their employees with mobile solutions. Different from some views in the literature of the 1990s (e.g. (Picco, 1998)), communication costs are not the main reason for disconnected support and thus not the main obstacle for mobile applications anymore. A number of already existing information systems would gain from mobile support, but due to lacks in network connectivity, mobile information systems cannot be available anywhere at any time. To increase the benefit of those applications, the next logical step is to provide support for an offline modethat allows to continuously work with an application, even when the device is disconnected from a network.

We have shown that some problems emerging from hybrid support and replicating data can be overcome quite easily, while others pose more difficult questions. As (Jørstad et al., 2004a) and (Jørstad et al., 2004b) propose service continuity for generic services, some of their proposed components are specifically designed to deal with problems arising from general and often even unknown services. Focusing on mobile information systems, those components appear to be superfluous while other components, namely monitor and I/O redirector are necessary. Implementing a connection manager that tackles the named problems and reduces the overhead for developing hybrid support for mobile information systems from the scratch is one of our current research projects; finding the most suitable way of integrating our connection manager into application architectures is another.

REFERENCES

Adl-Tabatabai, A.-R., Langdale, G., Lucco, S., and Wahbe, R. (1996). Efficient and language-independent mobile programs. In *Proceedings of the ACM SIGPLAN '96 Conference on Programming Language Design and Implementation (PLDI)*, pages 127–136.

Holder, O. (1998). The Design of the FarGo System (A Design Document). Technical Report EE Pub No. 1171, Technion - Israel Institute of Technology.

Joseph, A. D., deLespinasse, A. F., Tauber, J. A., Gifford, D. K., and Kasshoek, M. F. (1995). Rover: A Toolkit for Mobile Information Access. In *Proceedings of SIGOPS'95*, pages 156–171. ACM, ACM Press.

Jørstad, I., van Thanh, D., and Dustdar, S. (2004a). An analysis of service continuity in mobile services. *2nd International Workshop on Distributed and Mobile collaboration (DMC), WETICE conference*.

Jørstad, I., van Thanh, D., and Dustdar, S. (2004b). Towards Service Continuity for Generic Mobile Services. *The 2004 IFIP International Conference on Intelligence in Communication Systems (INTELLCOMM 04)*.

Julien, C. and Roman, G.-C. (2002). Egocentric context-aware programming in ad hoc mobile environments. *Proceedings of SIGSOFFT 2002/FSE-10*.

Merriam-Webster (2003). *Merriam-Webster's Collegiate Dictionary*. Merriam-Webster.

Orfali, R., Harkey, D., and Edwards, J. (1996). *The Essential Client/Server Survival Guide*. Wiley Publ.

Pandya, R. (2000). *Mobile and personal communication systems and services*. IEEE Press.

Picco, G. P. (1998). μCODE: A Lightweight and Flexible Mobile Code Toolkit. *Lecture Notes in Computer Science*, 1477:160–171.

Roth, J. (2002). *Mobile Computing*. dpunkt.verlag.

Starobinski, D., Trachtenberg, A., and Agarwal, S. (2003). Efficient PDA Synchronization. *IEEE Transactions on Mobile Computing*, 2(1):40–51.

Weinberg, Y. and Ben-Shaul, I. (2002). A Programming Model and System Support for Disconnected-Aware Applications on Resource-Constrained Devices. In *Proc. of the 24th Int. Conf. on Software Engineering (ICSE'02)*, pages 374–384. ACM Press.

Weiser, M. (1991). The computer for the twenty-first century. *Scientific American*, 265(3):94–104.