Patterns of Mobile Interaction

Jörg Roth

University of Hagen, 58084 Hagen, Germany

Abstract: The design of systems for mobile scenarios covers a wide range of issues, ranging from mobile networking to user interface design for mobile devices. Mobile applications often run distributed on several connected devices, used by many users simultaneously. Considering all issues related to mobile scenarios, a designer might be overwhelmed. As a solution, we propose a specific kind of design patterns which we call *mobility patterns*, derived from successful mobile applications. They allow a designer to re-use design elements as building blocks in their own designs. After describing the idea of mobility patterns, we give a brief overview of patterns we have identified so far. Two patterns are described in more detail with the help of our research platforms *QuickStep* and *Pocket DreamTeam*.

Keywords: Application design; Design patterns; Mobile computing; Mobile interaction

1. Introduction

Mobile computing fundamentally differs from desktop computing. Compared to desktop computers, mobile devices (e.g. PDAs, mobile phones and digital cameras) have low computational power, small memory and often no mass storage. Communication links to other mobile devices or to a stationary network are usually wireless, and thus are often unstable with low bandwidth.

These aspects highly influence how users interact with mobile systems. Often, users interacting with a mobile system interact with other users (e.g. using mobile phones). In contrast to traditional applications, users often interact with more than one computer or device at the same time. For example, in using a PDA for browsing the Internet, a user may have to set up a wireless connection using a mobile phone.

2. Patterns

Having several users and several devices in a system design, a designer has to consider several problem areas. To give some examples:

- A single application has to be developed in a distributed manner, i.e. parts have to be identified which run independently on different devices.
- Due to the poor availability, bandwidth and reliability of mobile connections, we have to solve network problems.
- Security (e.g. privacy, integrity and authenticity) is an important issue. Data in mobile scenarios are often confidential [1], but wireless communication is very insecure.
- When designing user interfaces for mobile devices, especially for heterogeneous envinonments, we have to consider the special user requirements as well as the capabilities of the devices involved [1,2].

Often, a design focuses on a specific problem area and neglects others. Especially in the complex field of mobile computing, neglecting problems can cause an entire design to fail. On the other hand, taking into account all the issues in early design stages may overwhelm a designer or a design team.

As a possible solution, we propose a tool that can be successfully used in different scenarios: *design patterns*. Patterns are derived from successful software designs and can be re-used as building blocks for new designs. In our approach, we use a specific kind of design patterns we call *mobility patterns*. Mobility patterns cover problem areas we find very often in mobile scenarios. We grouped related patterns to *pattern classes* using a pattern hierarchy (Fig. 1). The white boxes represent the patterns, the grey boxes represent pattern classes.

Mobility patterns are not only applicable for mobile scenarios, but some patterns appear very often in mobile application projects, thus they are good candidates for new projects.

Personal and Ubiquitous Computing (2002) 6:282–289

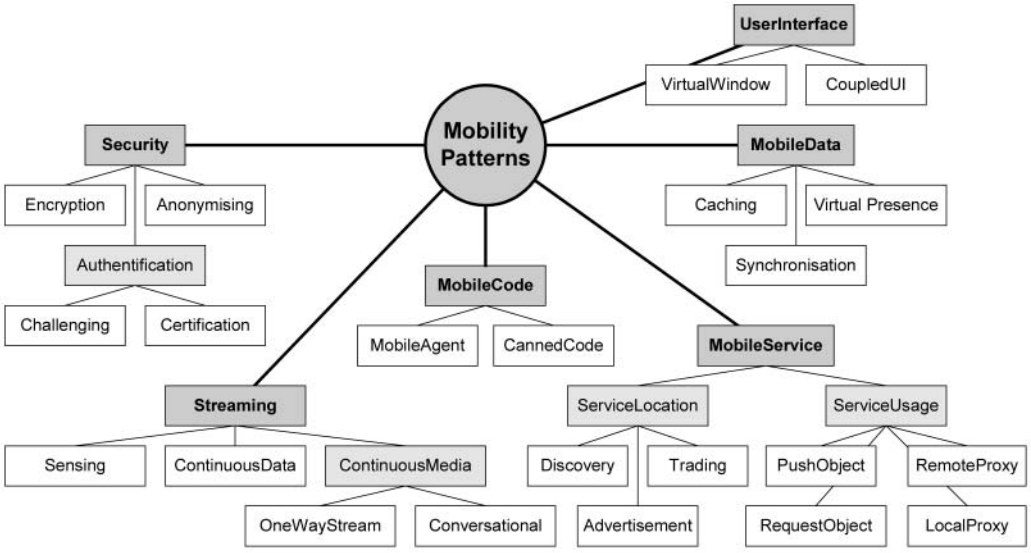


Fig. 1. The pattern hierarchy.

3. Example Patterns

To describe patterns, we followed established description formats [3]. Each description contains the sections: *Pattern Name*, *Synopsis*, *Context*, *Forces*, *Solution*, *Consequences*, *Examples*, *Related Patterns* and *Classes*. From the proposed description used in object-oriented software development, we replaced the section *Implementation* by *Examples*. It is difficult to give a tangible implementation for a specific mobility pattern – a list of good examples is more useful. In addition, we added the section *Classes*, which indicates how to integrate a pattern into the pattern hierarchy. Pattern classes offer additional structuring information to the designer, thus problems and solutions related to a specific pattern can be classified more easily. Compared to more formal approaches [4], mobility patterns have a strongly informal characteristic. To give an impression of mobility patterns, we present two patterns: *Synchronisation* and *Remote Proxy*.

3.1. Example 1: The synchronisation pattern

Synopsis: identical data is stored on different devices or computers, which are weakly connected. Several users apply data changes to different devices, often simultaneously.

Context: consider two users carrying around databases stored in their PDAs, originally copied from a central server. While travelling, they are only rarely connected to their home database.

Forces: identical data stored at different locations tends to run *out of sync*, when users apply modifications on their locally stored data and device are only rarely connected.

Solution: each device or computer provides a *sync engine* (Fig. 2).

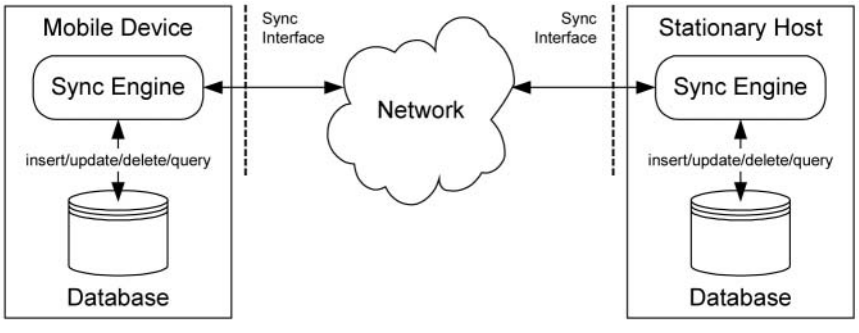


Fig. 2. Communication in the synchronisation pattern.

Each sync engine

- keeps track of modifications applied to local data;
- exchanges modifications with corresponding databases on other devices whenever they reconnect; and
- detects conflicts and realises a conflict resolution strategy.

Sync engines communicate with each other via sync interfaces. Note that Fig. 2 only shows a sample scenario; two mobile devices could also synchronise their data.

Consequences: we can use this pattern whenever data stored on different devices has not to be strongly up to date. However, users should be aware that other users could change the same data simultaneously, which may cause conflicts later.

Since connected devices have to effectively compare their local changes, only data that can be stored in tables or lists is suitable for this pattern. Ideally, changes are logged for every field in a record. Weakly structured data must be compared record by record, which increases the probability for unwanted conflicts. This pattern is not suitable for continuous data such as audio or video.

Examples: SyncML (http://www.syncml.org) is a framework for data synchronisation in mobile scenarios. Palms Hotsync allows a user to synchronise a palm device with one or more desktop PCs. We present another example, the QuickStep platform, below.

Related Patterns: VirtualPresence

Classes: MobileData

3.2. Example 2: The remote proxy pattern

Synopsis: a device does not have the capability to perform a requested task. It connects to another device with higher computational power, which acts as a delegate.

Context: consider a user browsing the web with a handheld device. The screen resolution of such a device is currently very poor, thus elements such as graphics and tables are difficult to display. In addition, rendering complex elements requires many computational resources, which are often unavailable on such a device.

Forces: a user who requests a specific task

- wants to save network bandwidth,
- wants to save computational resources (e.g. memory) on the local device, and
- expects appropriate input/output behaviour according to the locally available capabilities.

If the end-user's device itself offers network services to other computers, these computers are not interested in the specific device properties. In contrast, they expect the same behaviour (e.g. the same communication protocol) as if the device has full server capabilities. In particular, we do not want to change established protocols such as file transfer protocols or network file system protocols to access the mobile device.

Solution: the device does not connect directly to the network, but asks another device or computer to perform these tasks. This other computer, called the *proxy*,

- accepts service requests from other devices,
- connects to the actual service provider and performs the requested tasks,
- processes the results, and
- sends them back to the initiating device.

Figure 3(a) shows a traditional communication between one computer and a network. The network represents the set of involved corresponding hosts, i.e. those hosts that either offer services to the specific host or use services from the specific host.

Figure 3(b) shows the communication infrastructure according to the remote proxy pattern. The application is spread over two devices called the client and the proxy. We identified the following entities:

- The client is the end-user's device, thus the client must contain at least the user interface of the application. Other parts of the application may reside on the client due to performance considerations.
- The proxy runs the demanding parts of the application, e.g. heavy computation tasks. Usually, the proxy is a stationary host, which runs without user interaction. A user interface is only necessary for administration and configuration. A specific computer can be a proxy for more than one client. The relation between proxy and client may change dynamically.

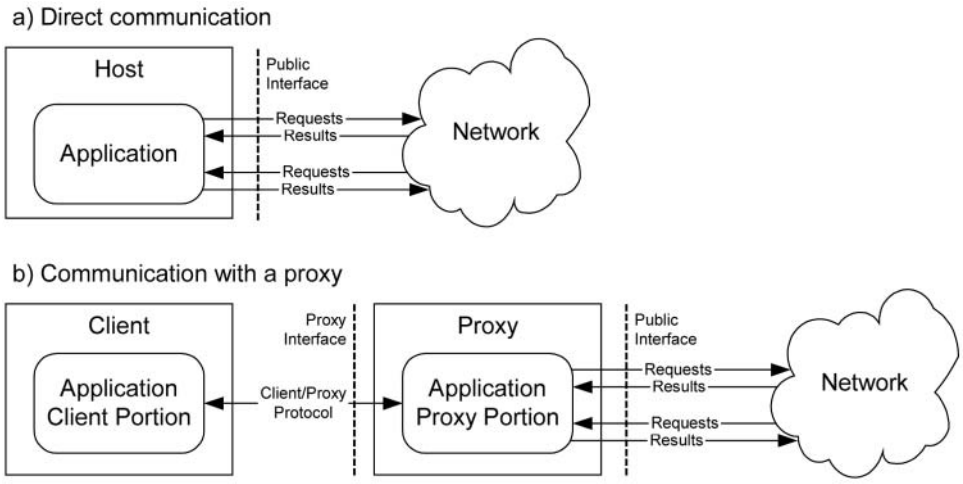


Fig. 3. Direct communication vs. remote proxy pattern.

- The public interface is the communication interface between proxy and network. The interface has to be identical to the interface presented in Fig. 3(a). From the network's view, it is not possible to distinguish the proxy architecture from a single device.
- The client accesses the proxy via the proxy interface. The interface has not to be public since other hosts of the network do not use it. As the client is often a mobile device and the proxy is stationary, the communication between client and proxy is usually wireless.

Consequences: this pattern is a general pattern and useful in various mobile scenarios. Very often, devices used by the end-users have only poor capabilities. Nevertheless, users want to execute demanding tasks. As the greatest benefit, we do not have to change the public interface. The rest of the network remains unmodified, i.e. we can use the same infrastructure and protocols. In this pattern however, there are two crucial points:

- The proxy itself: in the case of failure, task execution is disabled, even if the requesting device and the service provider are on-line.
- The communication link between the requesting device and proxy: if this link is broken, the task cannot be performed, even if the proxy has successfully executed the task. Cache mechanisms may be integrated into the client to reduce the effects of weak connections. If the client as well as the proxy maintain the same data (e.g. inside caches), the application has to provide concurrency control mechanisms to achieve consistent states of distributed data.

Obviously, the proxy must have more capabilities than the end-user's device. As the proxy provides a specific service, a mobile device must be able to find the proxy inside the network. This leads to the following consequences:

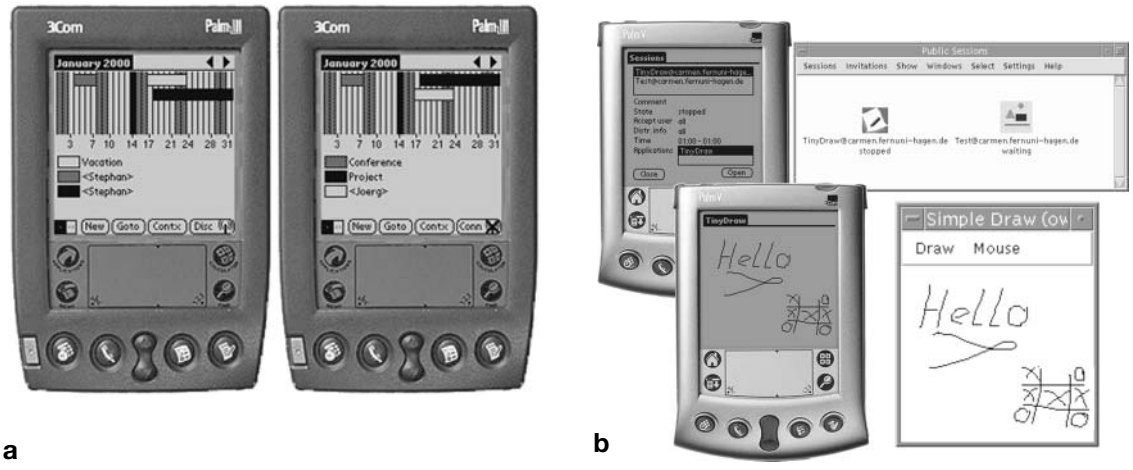
- A proxy must have a fixed network address or the mobile device can, with help of a service discovery mechanism, resolve the network address.
- The proxy has to be on-line whenever a mobile devices requests services.
- Currently, a proxy is usually a traditional workstation, not a mobile device.

Examples: Browse-it [5] allows a handheld user to browse the web without struggling with device limitations such as screen resolution. The proxy pre-processes web pages, downscales graphics and pre-computes the appropriate layout. As a result, the amount of data transferred to the handheld devices is drastically reduced, and the devices are relieved from heavy rendering tasks.

Another example, Pocket DreamTeam, is presented below.

Related Patterns: LocalProxy, PushObject, RequestObject.

Classes: ServiceUsage, MobileService.



a b

Fig. 4. (**a**) QuickStep and (**b**)DreamTeam.

4. Patterns in Research Platforms

The idea of mobility patterns heavily influenced our research platforms *QuickStep* and *Pocket DreamTeam*. Figure 4 shows screenshots of sample applications. Both platforms allow mobile users to share data among a group. For example, users could share dates and timetables (Fig. 4(a)) or edit collaboratively shared freehand sketches (Fig 4(b)). QuickStep uses the synchronisation pattern, Pocket DreamTeam is based on the remote proxy pattern. The different patterns lead to completely different architectures and thus to different platform characteristics.

4.1. QuickStep

The QuickStep platform [1,6] (Fig. 4(a)) supports developers of mobility-aware collaborative handheld applications. They can use communication, collaboration and dialogue primitives provided by the platform, and can concentrate on application-specific details. A developer can integrate predefined awareness widgets into an application with a few lines of code. We can summarise the QuickStep approach as follows:

- QuickStep supports applications with well-structured, record oriented data, as being used by built-in software for handheld devices (e.g. for to-do lists, memos, telephone lists). QuickStep was explicitly not designed for supporting multimedia data, graphical oriented applications or continuous data streams.
- QuickStep mainly supports synchronous collaboration. We used the term *relaxed synchronous collaboration* to indicate that devices are not synchronised when they are disconnected.
- QuickStep provides awareness widgets for collaboration awareness as well as context awareness.
- QuickStep comes along with a generic server application, which supports arbitrary client applications without modifying or re-configuring the server.
- The QuickStep architecture ensures privacy of individual data.

Figure 5 presents the QuickStep architecture.

The computational power of handhelds and the network bandwidths are very low as compared with desktop environments. To reduce network traffic and to perform as many computations as possible on a server, we developed a combined mirroring and caching mechanism that we designed according to the synchronisation pattern. Each handheld has its own *local database*, which contains the records of the application. Only the owner can add, change or remove local records. The QuickStep server has a copy of each local database, the *mirror database*. The mirror databases are incrementally updated each time a handheld device re-connects to the server. To view data when mobile devices are disconnected, a local copy of other users' mirror database exists on the handhelds, called the *cache database*. Since the amount of data of all mirror databases might be too big for the handheld, a selector set by the application reduces the number of cache entries.

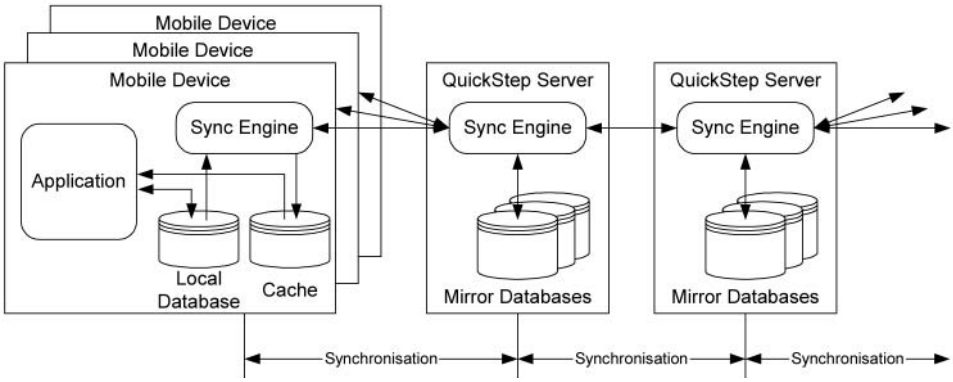


Fig. 5. The QuickStep architecture.

Each server stores the local data of mobile user in wireless communication range (e.g. infrared or WLAN range). To cover a wider area, servers can be connected. We applied the synchronisation pattern to each 'hop', e.g. between handheld and server and between two servers. Whenever a mobile user modifies the local database, this change will eventually reach all other mobile devices in the covered area.

4.2. Pocket DreamTeam

Pocket DreamTeam [7] (Fig. 4(b)) is the PalmOS version of our groupware platform DreamTeam [8]. The DreamTeam environment allows the developer to develop synchronous collaborative applications (e.g. collaborative diagram tools, text editors, shared web browsers) like single user applications, without struggling with network details or synchronisation algorithms. DreamTeam was originally developed for PCs or workstations running Java. It is based on a fully decentralized architecture without the need for a central server. Shared data are distributed among the group members using an automatic replication mechanism.

When we finished the implementation of the desktop variant of DreamTeam, we planned to develop a handheld version, which should meet the following requirements:

- The handheld version and the desktop version of DreamTeam should run inside the same network. It should be possible to form arbitrary sessions of handheld and desktop users, e.g. with only handheld users, only desktop users or a mixture.
- The original desktop variant of DreamTeam should still run *without any changes*. The DreamTeam runtime system contains approx. 125,000 lines of Java code. About 20 collaborative applications have been developed so far. Any modifications of the original DreamTeam core were not under discussion.
- We did not have the goal to run original DreamTeam applications on handheld devices. Since handheld devices differ fundamentally from desktop computers, it is not reasonable to follow the desktop usage paradigms based on, for example, overlapping windows with graphics. We accepted the reimplementation of some parts of DreamTeam for handheld devices. However, we wanted to keep the amount of new developments as small as possible.

Since PalmOS does not provide some necessary services to run the full DreamTeam platform, Pocket DreamTeam uses the remote proxy pattern. Since most of the work is done by the (desktop) proxy application, the Pocket DreamTeam program is very small (some Kbytes) compared to the original DreamTeam platform (some hundreds Kbytes).

Figure 6 presents the Pocket DreamTeam architecture. The boxes on the right-hand side represent desktop DreamTeam systems. They remain unmodified and use the traditional DreamTeam protocol to communicate to other systems, including the mobile ones. We divide the overall network into two segments: a mobile segment, which contains the mobile device and the wireless connection, and a stationary segment, which contains traditional DreamTeam systems, the proxy and the core network.

Since the proxy resides in the stationary segment, it can act as a placeholder whenever the mobile device gets disconnected. Moreover,

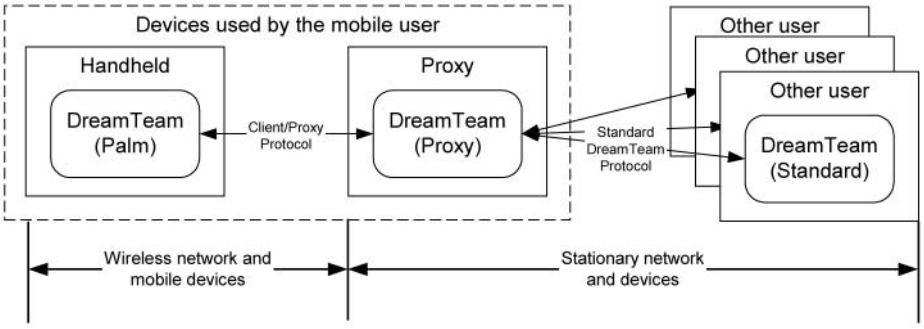


Fig. 6. The DreamTeam architecture.

it hosts heavy computation tasks, which the mobile device is not able to run.

4.3. Comparison

QuickStep and Pocket DreamTeam are typical platforms based on mobility patterns. Mobility patterns strongly influence a design of the platforms on an informal level, and do not restrict the design process.

The remote proxy pattern provides an architectural decomposition of applications, and is especially suitable for mobile devices with low computational power. Mobile devices can transfer heavy tasks to stationary proxies, which act as a placeholder to the network. As the major disadvantage, we identified the communication link between client and proxy, which is usually a wireless connection with poor network performance.

The synchronisation pattern, on the other hand, is especially suitable for weak connected devices. While they are disconnected, mobile devices have to store a considerable amount of data locally. In addition, we have to develop a sync engine for all involved devices and computers. As an advantage, mobile applications according to the synchronisation pattern can run independently from stationary hosts for a considerable time.

5. More Mobility Patterns

Table 1 lists some more mobility patterns we have identified so far. We provide only a brief synopsis and an overview of examples.

6. Benefits

Using mobility patterns has many advantages. Primarily, patterns are a tool to describe designs. Using the proposed pattern names, a designer can precisely express which building blocks are used for a specific system, thus misunderstanding is more unlikely.

As described above, mobile applications cover a wide range of issues. Designers tend to 'abstract away' or forget consequences of specific design aspects, thus possibly build systems which are not runnable. As a second benefit, patterns come along with a list of implications and consequences. A designer knows the pros and cons of a specific pattern.

As a third benefit, patterns allow a designer to

Table 1. A list of more mobility patterns

Pattern	Synopsis	Examples
VirtualPresence	Pieces of data are virtually present to other devices and can be accessed or modified according to the access rules of the host.	Windows CE remote file access, Coda [9]
RequestObject	A device requests a specific object (e.g. a web page) from another device.	WAP
PushObject	A device sends a specific object without request.	SMS, OBEX
LocalProxy	A *local* instance provides an interface to a local or remote service.	Locally running web proxy
OneWayStream	One-way transmission of audio or video data.	Video on demand, UMTS
Conversational	Two-way transmission of audio or video data.	GSM, DECT, Bluetooth audio
VirtualWindow	A device presenting a window or desktop of another device or computer.	Pebbles [10], PalmVNC
CannedCode	A device sends code, which is executed on another device. The code has not to be executable on the sender's device.	WMLscript, web filters
Sensing	A device receives continuous sensor data (e.g. location) from other devices.	GPS

re-use successful designs. Re-using software on lower levels (e.g. with software components) fails due to the heterogeneity of the networks and devices involved. With mobility patterns, we can re-use at least the design of a successful application.

The current pattern hierarchy does not claim to be complete. In future, we want to complete our collection of mobility patterns. For this, we analyse existing mobile computing applications and frameworks.

References

1. Roth J, Unger C. Using handheld devices in synchronous collaborative scenarios. 2nd international symposion on handheld and ubiquitous computing (HUC2K), Bristol, UK (Lecture Notes in Computer Science 1927), Springer-Verlag, 2000; 187–199
2. Calvary G, Coutaz J, Thevenin D. A unifying reference framework for the development of plastic user interfaces. 8th IFIP working konference on engineering for human-computer interaction (EHCI'01), Toronto, Canada (Lecture Notes in Computer Science 2254), Springer-Verlag, 2001; 173–192
3. Gamma E, Helm R, Johnson R, Vlissides J. Design patterns: elements of reusable object-oriented software. Addison-Wesley, 1995
4. Borchers J. A pattern approach to interaction design. Conference proceedings on designing interactive systems: processes, practices, methods, and techniques. Brooklyn, NY, 2000; 369–378
5. Pumatech, Browse-it for Palm Computing Platform User Guide, Puma Technology, Inc., San Jose, CA, 2000
6. Roth J. Information sharing with handheld appliances. 8th IFIP working conference on engineering for human-computer interaction (EHCI'01), Toronto, Canada (Lecture Notes in Computer Science 2254), Springer-Verlag, 2001; 263–279
7. Roth J. Mobility support for replicated real-time applications, innovative internet computing systems (I2CS), Kühlungsborn, Germany (Lecture Notes in Computer Science 2346), Springer-Verlag, 2002;181–192
8. Roth J. DreamTeam – A platform for synchronous collaborative applications. AI & Society 2000; 14(1): 98–119
9. Kistler JJ, Satyanarayana M. Disconnected operation in the Coda file system. ACM Transactions on Computer Systems 1992; 10(1): 3–25
10. Myers BA, Stiel H, Gargiulo R. Collaboration using multiple PDAs connected to a PC. Proceedings ACM 1998 conference on computer supported cooperative work 1998; 285–294

Correspondence to: J. Roth, Department of Computer Science, University of Hagen, 58084 Hagen, Germany. Email: Joerg.Roth@fernuni-hagen.de