Information sharing with handheld appliances

Jörg Roth

University of Hagen, Department for Computer Science
58084 Hagen, Germany
Joerg.Roth@Fernuni-hagen.de

Abstract. Handheld appliances such as PDAs, organisers or electronic pens are currently very popular. Handhelds are used to enter and retrieve useful information, e.g., dates, to do lists, memos and addresses. They are viewed as standalone devices and are usually not connected to other handhelds, thus sharing data between two handhelds is very difficult. Rudimentary infrastructures to exchange data between handhelds exist, but they are not designed for a seamless integration into handheld applications. The fundamental different nature of handheld devices to desktop computers leads to a number of issues. In this paper, we first analyse the specific characteristics of handheld devices, the corresponding applications and how users interact with handhelds. We identify three basic requirements for a successful realisation: privacy, awareness and usability. Based on these considerations, we present our own approach.

1 Introduction

Currently, there is a growing market for handheld devices such as PDAs, mobile phones, electronic pens etc. Upcoming communication technologies like UMTS and Bluetooth promise new functionality to support people to communicate which each other. Currently, the most accepted way for communication is still verbal communication. Symbolic or textual media such as SMS or Email are inconveniently to use in connection with handheld devices.

Handheld devices already contain a huge amount of data, e.g., appointments, holidays, addresses and agendas. It should be possible to exchange these data between users without a considerable effort. On the other hand, handheld devices basically have a private nature. Personal data are stored on such devices, which usually should not be shared with other people. This contradictory characteristics may be one reason, why handhelds are currently viewed as autonomous systems without any communication abilities to other handhelds.

In this paper we present the requirements, issues and problems of communication-oriented, distributed applications for handheld appliances. A lot of research was done about the design of distributed applications in desktop environments. However, due to the fundamental different nature of handheld devices, this knowledge can hardly be adapted to handheld scenarios.

This paper is structured as follows: first we present the fundamental differences of handheld devices to traditional desktop computers. Based on these considerations we

identify three key requirements, a communication-oriented distributed handheld application has to meet: *privacy*, *awareness* and *usability*. We then present a framework which allows to create successful information sharing handheld applications.

2 Handheld computing characteristics

The notion of *handheld device*, *palmtop*, *PDA* and *organiser* is often interpreted in different ways. One older interpretation distinguishes between *pen-based devices* and *palmtops*, where the latter have keyboards. In contrast, Microsoft divides Windows CE devices into *handheld PCs* (H/PC) with a keyboard, *palmsize PCs* (P/PC) which are controlled by pen and *handheld PC Pro* devices, which are subnotebooks [3]. To get completely confused, Microsoft calls the new pen-based devices based on Windows CE 3.0 *Pocket PC*.

In the following, we understand by *handheld devices* mobile devices with small displays, without or only with rudimentary keyboards and an autonomous power supply. Examples are:

- 3Com's Palm devices (e.g. Palm III or Palm V),
- Casio's Cassiopeia or
- Electronic pens such as the C-Pen.

In particular, we do not summarise notebooks or laptops under the notion of handhelds.

Handheld computing is closely related to so-called *ubiquitous computing*. Mark Weiser introduced the concept of ubiquitous computing often called "ubicomp" [19]. Weiser's vision was a huge number of invisible and "calm" computers surrounding people in their everyday life. In contrast, handheld computing keeps the device in the foreground, where ubicomp devices should work in the background [6]. People using handhelds are aware of using computers and adapt their activities to the device, e.g. learn a specific kind of handwriting. However, studying handheld computing may be the right step towards ubicomp, since a number of problems are identical. E.g., handhelds should be suitable for everyday tasks, easy to handle and failproof.

Hardware and software characteristics

Table 1 shows some hardware characteristics of popular handheld devices.

Table 1. Some hardware characteristics of handheld devices

Device	Processor	RAM	Screen	Battery life
Palm IIIxe (3Com)	16Mhz MC68EZ328	8MB	160x160 (16 grey)	1.5-2 months (normal use)
Cassiopeia E-15 (Casio)	69MHz NEC VR4111	16MB	240x320 (16 grey)	25 hours (continuously)
C-Pen 800 (C-Technologies)	100Mhz Intel StrongARM	8MB	200x56 (b/w)	2-3 weeks (normal use)

Compared to desktop computers, handheld devices have small memory, low computational power, limited input and output facilities and usually no mass storage. Having no mass storage, persistence data have to be stored in the battery-buffered RAM, what decreases the available RAM during runtime for dynamic data and runtime stack. Even worse: some CPUs (e.g. Palms CPU) allow only to address small pieces of memory (64k) as a whole. The reduced capabilities have two major reasons:

1. Size: The display is limited to the area which conveniently can be hold in one hand (approx. 10cm x 8cm). Chips (e.g. memory and CPU), even when highly integrated, need space for connectors and circuit boards. Due to these limitations, it is impossible or at least cost intensive to integrate high resolution displays or a big number of electronic parts inside a device.

2. Battery life: Fast CPUs, big memories and high resolution displays (particularly coloured) cost a big amount of valuable battery power. If battery technology will not be significantly improved in the future, handheld computers will always be far behind the capabilities of desktop computers.

The reduced equipment of handheld devices has big influence on software development. Handheld applications are usually not developed 'from scratch'. Handhelds come along with their own operating system, e.g. PalmOS [2], Windows CE [3], EPOC [17] or ARIPOS [5]. These handheld operating systems cover the following area of services:

- starting, stopping and switching applications, memory management and managing the user interface;
- special device-dependent services like handwriting recognition, OCR, performing time-dependent alarms;
- supervising the battery power, performing auto-power-off;
- managing persistent data in the battery-buffered memory;
- managing communication to other devices (usually to the host PC).

Software development kits allow to code and compile handheld applications on desktop computers, usually in C. Some kits allow to test applications on emulators before they are downloaded on the specific device.

Compared to desktop operating systems, handheld operating systems do not offer the same variety of services. The major shortcomings are:

Limited user interface capabilities: Usually the so-called WIMP paradigm (Windows, Icons, Menus, Pointers) which is very common on desktop computers, is not supported on handhelds or in a very reduced manner. This is because of the limited display and input capabilities.

Limited support for persistent data: Since handhelds have no mass storage system, all persistent data have to be kept in the battery-buffer RAM. Windows CE emulates a hierarchical file system inside the RAM area. Other systems like ARIPOS and PalmOS store persistent data in so-called *databases* [2] (not to be confused with traditional databases). A database is a persistent collection of records. Each record has a unique identifier; its content is opaque to the operating system. Constructing and interpreting records solely depends on the corresponding application. Databases can be viewed as flat file system with record-oriented structure.

Limited or no parallel execution capabilities: Most handheld operating systems do not support threads or processes for background tasks, which is a common technique

for desktop computer applications. As a work-around, some systems offer so-called *timers* which can periodically call a predefined procedure. Unfortunately a call is only performed, when no other instruction is being executed, thus a timer does not provide real background operations.

Limited support for communication: Handhelds only support a small set of communication capabilities compared to desktop computers. Handheld operating systems usually support one specific way of communication determined by the peripheral equipment. E.g., ARIPOS only supports IrDA communication, since the C-Pen only has an infrared transceiver to communicate. PalmOS supports serial communication and TCP/IP, but does not allow to install a TCP server socket. Server sockets are essential to react on incoming communication requests.

Networking

In addition to the limited communication support, the underlying network itself has some drawbacks:

- Wireless communication infrastructures currently have low bandwidth (GMS, e.g., only provides 9600 Baud) and tend to have high error rates and abnormal terminations.
- Handhelds as communication end points are mobile in the network, i.e. often change their network addresses.
- Mobile devices are rarely connected, i.e. are most of the time not available in the network because of network failures or just because the device is turned off.

Emerging technologies like UMTS, IPv6 and Bluetooth will change the way handhelds are used inside a network. E.g., UMTS allows a device to be permanently connected to the network with high bandwidth. IPv6 offers with MobileIP the possibility to have always the same IP address, even when a device moves inside the network. However, such technologies are not yet widely available and cannot be used inside current concepts.

All topics mentioned above have big influence on the entire application development process. Usually, developing handheld applications is very cost intensive as a result of the limited handheld capabilities. This includes testing and debugging, since handhelds offer not the same debugging facilities than desktop computers.

3 Handheld applications

Interaction with handheld applications follows a different usage paradigm as interaction with desktop applications. First, handheld applications have to respect the limitations mentioned above. Much more important: users require another availability of such applications: handhelds do not 'boot up'. Applications have to immediately appear on screen. In turn, when the handheld device is deactivated, an application has to immediately save its state. In general, handheld applications are developed to enter and retrieve small pieces of information rather than processing data.

Application types
On handhelds, only a small set of application types is reasonable. To get a deeper insight into this topic we analysed a set of 32 Palm applications currently available as shareware. In order to get a representative set of applications, we took a shareware collection of a popular german journal [4]. Table 2 shows the results.

Table 2. A selection of Palm applications

Applications	**Application type**	**Data type**	**Count**
Launcher III, SwitchHack, German Chars, Hackmaster, Eco Hack	Utilities	mixed	5
Brainforest, dNote, HandyShopper, PocketMoney, HanDBase	Textual notes, ideas, shopping lists, bank accounts	textual documents, tables	5
Feiertage, Yearly, DateBk3, Palm Planner	Dates, appointments, holidays	dates	4
Abacus, TinySheet, MiniCalc	Spreadsheet tools	spreadsheets	3
Desktop to Go, Documents to Go, TealDoc	Documents	textual documents	3
ptelnet, MultiMail, HandWeb	Internet tools	mails, web pages	3
DiddleBug, TealPaint	Graphical notes, freehand	graphical data	2
Parens, Currency Calculator	Calculators	numbers	2
PocketChess, TetrisV	Games	game states	2
Secret!	Security	texts	1
Route Europe	Route planner	geographic data	1
Timer	Clock	time	1

Five applications are utilities and so-called *hacks* which extend the operating system. This kind of application is not used to store information, thus not taken into further considerations.

Except for two programs which allow graphical input, all applications store well structured and record oriented data, often text based. Typical data of handheld applications are

- texts and lists of texts,
- date entries,
- numbers and
- tables or spreadsheets.

None of the applications above deal with multimedia data such as audio or video. Audio and video require a considerable network bandwidth, sufficient output devices and a huge amount of battery charge. Currently multimedia data is not suitable for handheld devices.

To summarise, most of the application deal with simple data types such as strings or numbers, joined together as lists or tables. Only few applications have a graphical nature.

Privacy

Data stored inside a handheld device are usually viewed as *private*. Even more than desktop computers, such devices are viewed as personal ones [16]. Personal data, e.g. telephone numbers, birthdays and leisure-time activities are stored inside such a device. Some handheld operating systems protect private data. E.g., PalmOS allows to mark some entries as *private*. Such entries are only visible after a password is entered. In addition, the handheld device as a whole can be locked. After turning the device on, a password has to be entered.

Connecting handheld devices in order to exchange data increases the problem of private data. If a handheld device is connected to an untrusted network, applications have to offer mechanisms to guarantee privacy of individual data. No private data should be transferred across a network, other people should not have the possibility to break into a handheld in order to spy out data.

To gain acceptance by end-users, an infrastructure has not only to ensure privacy; an end-user has to be *convinced* that her or his private data is kept private. This is perhaps the most crucial issue related to privacy.

Awareness

Mobile devices can be connected to the network at different places. Depending on the location, different information is available. Users should be aware of their current location, including the geographic location as well as the location in the network. These information are part of the so-called *context awareness*. Abowd and Mynatt list different kinds of context awareness, defined by the "five W's": *Who*, *What*, *Where*, *When* and *Why* [1]. E.g., the *Who* context is based on information about other people in the environment, especially when looking at activities.

Sharing information between people leads to the area of groupware and CSCW (computer supported collaborative work). Collaborative applications significantly differ from single-user applications. Many users provide input (often simultaneously), output has to be processed for many users and shared data have to be kept consistent. Groupware applications have to provide a 'feeling' of working together in a group, called *collaboration awareness*: users have to be aware of other users involved in the collaborative task.

Context awareness as well as collaboration awareness require elements inside the applications' user interfaces called *awareness widgets*. Similar to users, we call an application *aware* of something, if it explicitly takes care for a special situation, otherwise we call it *transparent*. E.g., *collaboration aware* applications are especially designed to support a group, i.e. they contain special code for group functions. *Collaboration transparent* applications are original single-user applications, which, with help of a groupware toolkit, can be used by many users simultaneously. Collaboration transparent applications do not offer awareness widgets. A similar notion can be applied to the mobility aspect: *mobility aware* applications contain code to handle mobility, e.g. react on unstable network connections and changing network locations. *Mobility transparent* applications cannot handle such problems explicitly, but rely on an underlying platform.

Usability
Usable applications support users in carrying out their tasks efficiently and effectively. More than desktop applications handheld applications should have a high degree of usability. When an application is designed in isolation from the intended users, the result is all too often an application which does not meet their needs and which is rejected by end-users. An application should meet the following requirements:

Respect hardware and software limitations: Usable applications consider the hardware and software of the hosting handheld device. This means, not to run heavy computation tasks on handhelds. User interfaces should be designed for small displays with minimal text input. Communicating across the network should consider the small bandwidth and high error rate.

Software quality: Handheld applications should be more failproof than desktop applications. A locked or crashed application blocks the entire device. An application hanging in an infinite loops prevents some devices from being switched off. Cold starting a handheld often results in loosing all stored data. The problem becomes even worse, if the applications communicates with other devices. One device being blocked may interrupt communication in the entire group. To improve software quality, design guidelines may help a developer to build well-formed applications. Such a guideline can, e.g., be found in [2]. A platform or application framework helps to fulfil these guidelines as it encapsulates standard solutions for a specific application domain. An application developer can rely on a set of services and only has to code application-specific functions.

Respect everyday requirements: Handheld applications are used every day. Abowd and Mynatt introduced the term called *everyday computing* [1]. They state that daily activities rarely have a clear beginning or end and often are being interrupted. This issue is especially important when considering communication-oriented applications. The strict classification between asynchronous and synchronous groupware [7], hardly applies to everyday tasks. This leads to the notion of *relaxed synchronous collaboration* when group members collaborate synchronously, but may infrequently disconnected from the network for short periods of time [15]. In addition, everyday tasks require spontaneous, unplanned communication. Exchanging data between handhelds should be as easy as a phone call. Especially, user-driven central co-ordination or administration should be avoided.

4 Related work

Mobile phones
Mobile phones offer simple mechanisms to transfer textual data. SMS (short message service) [13] is a protocol which allows to send up to 160 characters to another mobile phone. It can be slightly compared to the email service on the Internet, but is based on the mobile phone infrastructure GSM (global system for mobile communication). WAP (wireless application protocol) [20] allows to browse special Internet pages on small displays. WAP only provides a one-way information channel, i.e. it is not possible to send page contents from one device to another.

Beaming
A simple technique to exchange data between handhelds, so-called "beaming", comes along with the Palm device [2]. Manufacturer of other devices, e.g., of C-Pens or Windows CE devices adapted the technology. Beaming can be viewed as de-facto standard for short range data exchange between handheld devices. Beaming is based on Infrared and allows to exchange a single record of data, e.g. one address or one memo. Involved devices should be inside a range of approx. one meter. The action of beaming is done manually, i.e. the sender as well as the receiver have to interact with their device, every time an entry is transferred. Beaming is only suitable for a small amount of data.

PIMs
Personal Information Managers (PIMs) are important tools when using handhelds. PIMs conveniently allow to enter a data by keyboard and then download it to the handheld device. A popular PIM is Microsoft's Outlook [11]. In addition to synchronising data with a handheld, Outlook allows to schedule appointments in a team, thus it is possible to exchange limited data between handhelds among a group of people.

Coda
Several research platform have been developed to address the problem of data distribution and consistency in mobile environments. Coda [9] provides a distributed file system similar to NFS, but allows disconnected operations. Applications based on Coda are fully mobility transparent, i.e. run inside a mobile environment without any modification. Disconnected mobile nodes have access to remote files via a cache. Operations on files are logged and automatically applied to the server when the client reconnects. Coda applications can either define themselves mechanisms for detecting and resolving conflicts or ask the user in case of conflicts.

Rover
The Rover platform [8] supports mobility transparent as well as mobility aware applications. To run without modification, network-based applications such as Web browsers and news readers can use network proxies. The development of mobility aware applications is supported by two mechanisms: *relocated dynamic objects* (*RDOs*) and *queued remote procedure calls* (*QRPC*). RDOs contain mobile code and data and can reside on a server as well as on a mobile node. During disconnection, QRPCs are applied to cached RDOs. As in Coda, operations are logged and applied to server data after reconnecting.

Bayou
Bayou [18] provides data distribution with the help of a number of servers, thus segmented networks can be handled. In contrast to Coda, replicated records are still accessible, even when conflicts have been detected but not resolved. Bayou applications have to provide a conflict detection and resolution mechanism. Ideally, no user intervention is necessary. Bayou is not designed to support real-time applications.

Sync
Sync [12] allows asynchronous collaboration between mobile users. Sync provides a collaboration based on shared objects which can be derived from a Java library. As in Bayou, data conflicts are handled by the application. Sync applications have to provide a *merge matrix*, which contains a resulting operation for each pair of possible conflicting operations. With the help of the merge matrix, conflicts can be resolved automatically.

Lotus Notes
Lotus Notes [10] has not primarily been designed for mobile computers, but allows replicated data management in heterogeneous networks. Nodes can be disconnected and merge their data after reconnection. Data in Lotus Notes have a record structure. Fields may contain arbitrary data which are transparent to Notes. Records can be read or changed on different nodes simultaneously. When reconnecting, conflicting updates are resolved by users. With help of the Notes extension Mobile Notes, it is possible to access databases via Palm devices and mobile phones.

Discussion
Mobile phone protocols are designed for very simple data and not practical for structured data. It is difficult to adapt application specific data with an internal record structure to these protocols. A good solution for small amounts of data provides beaming, since any application can use this communication mechanism to exchange records with other handhelds. Due to the record-by-record character, beaming is not suitable for a reasonable amount of data.

Outlook is a solution especially designed for office environments and can hardly be adapted to other everyday tasks. It is not possible to add new applications to Outlook. In addition, Outlook requires a considerable amount of central administration.

Most of the research toolkits above request their mobile clients to be notebook computers with, e.g., hard disks. The focus of these platforms is to maintain data consistency in a weakly connected environment. Problems related to handheld devices such as small memory and reduced computational power are not handled satisfactorily. Automatic conflict detection and resolution need a considerable amount of resources on the handheld devices. We believe that such mechanisms are (currently) not suitable for handheld scenarios.

Concepts, such as the Rover toolkit which require mobile code and marshalling/unmarshalling mechanisms currently cannot be adapted to handheld devices, since they are significantly different from their servers. The concept of mobile code requires platform independent code and identical runtime libraries on both platforms. Even though languages such as Java are running on many platforms, handheld portings will provide other runtime libraries, thus mobile code mechanisms will fail.

The platforms above left many problems described above unsolved. Especially privacy and awareness are open issues.

5 The QuickStep approach

The QuickStep platform [15] allows to develop mobility aware and communication-oriented handheld applications. Developers can use communication and collaboration primitives provided by the platform and can concentrate on application-specific details. A set of predefined awareness widgets can be integrated into an application with a few lines of code. The QuickStep approach can be described as follows:

- QuickStep supports applications with well-structured, record oriented data. It has explicitly not been designed for supporting multimedia data, graphical oriented applications or continuous data streams.
- QuickStep provides awareness widgets for collaboration awareness as well as context awareness.
- QuickStep applications are fully collaboration and mobility aware.
- QuickStep comes along with a generic server application which allows to support arbitrary client applications without modifying or reconfiguring the server.
- The QuickStep architecture ensures privacy of individual data.

Before describing the QuickStep platform itself, we present two sample applications developed with QuickStep.

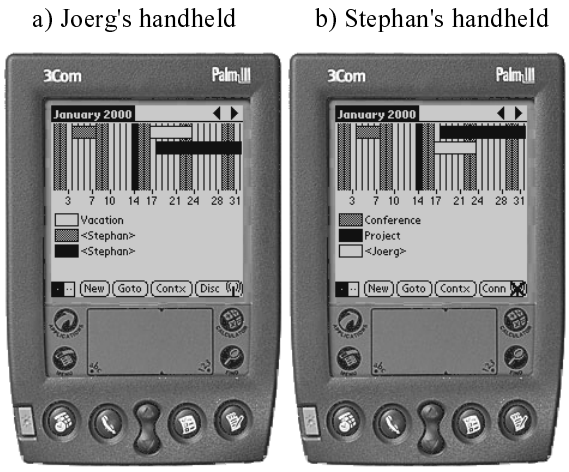

Fig. 1. A collaborative calendar tool

5.1 Sample applications

The first sample application allows a group of users to exchange date information (e.g. of vacations or travellings). This tool is useful in meetings, in which members want to schedule appointments for future meetings. Each member owns a handheld device, which already contains a list of appointments as well as entries indicating the

time one is unavailable. The problem is to find a date, when all members are available. Figure 1 presents an application that can help to find such a date.
The figure shows the view of two users on their personal handheld device. The upper half of the window displays the days of a month. Each range of dates when someone is unavailable is indicated by a bar. To get a better overview, the view can be switched to a two-months display. The lower half of the window is the legend for the upper half.

The two users Joerg and Stephan can see their own bars and the bars of each other. Foreign bars are labelled by the user name rather than the local label. For other users only the date range is of interest, not why someone is unavailable. Each user can make new entries which are distributed to the other user in real-time. With the help of this application it is very easy to find dates, where all members are available.

Fig. 2. A business card collector

The second example, the business card collector (figure 2), is a useful application for conferences. The application shows a list of all users assembled at a specific location. A user can view these cards and collect interesting cards in a persistent area. If the user permits, the business card collector publishes the card automatically, when entering a location.

To develop such an application 'from-scratch', a developer has to implement many tasks, e.g., communication protocols have to be integrated, shared data have to be managed. The application should offer awareness widgets. All these services have to be developed in addition to the main task. Developing all these functions would overwhelm a developer. QuickStep helps a developer to concentrate on the application-specific details. Data primitives as well as predefined awareness widgets can be used from the platform.

In the following, we present the QuickStep platform. After describing the basic concepts, we discuss QuickStep with help of the three key requirements *privacy*, *awareness* and *usability*.

5.2 The QuickStep infrastructure

As described above, handheld operating systems offer only limited support for communication. Most systems cannot handle communication in the background. If the handheld device is always the initiating part of the communication, we need an additional computer, which acts as a communication relay between handhelds. This computer, the *QuickStep server*, contains a generic server application which is able to serve arbitrary QuickStep applications.

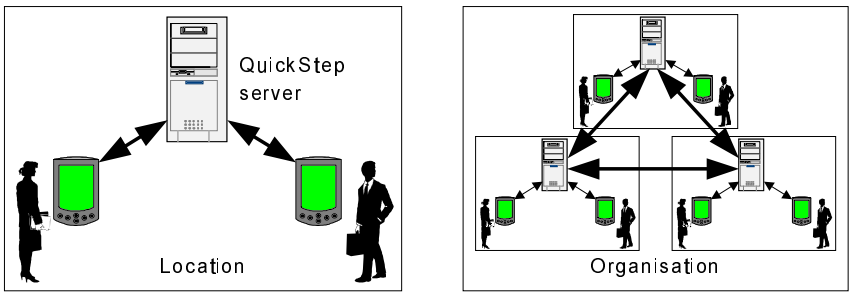

Fig. 3. QuickStep communication infrastructure

Figure 3 shows the QuickStep communication infrastructure. A QuickStep server operates in so-called *locations*. A location links all handheld devices together which are 'in range', i.e. which can be accessed by the specific communication technology. This can be the range of an infrared transceiver or a Bluetooth sender. Many locations linked together form an *organisation*. Organisations connect those locations which are in the same company, building, conference or public place. Table 3 shows typical examples for locations and organisations.

Table 3. Examples for locations and organisations

	Train	**Company**	**Conference**
Location	wagon	meeting room, hallway	presentation room, foyer
Organisation	whole train	company building	whole conference

Connections between handhelds and QuickStep servers are usually wireless, where the QuickStep servers are connected among themselves via traditional local area networks. The QuickStep server can be viewed as 'inventory' of a specific location. Once installed, it normally has not to be reconfigured or administered. The server runs without an operator and does not need a user interface, thus can work invisibly behind a panel.

5.3 Underlying data

As mentioned above, most handheld operating systems offer an entity called *database* to handle application-specific data. The database is a common programming abstraction in handheld applications, thus the ideal abstraction for communication-oriented applications as well. QuickStep follows the same paradigm when collecting and distributing data. The QuickStep application programming interface (API) has similar database functions as the database API. An application developer can use well-known services to handle application specific data. Data stored in QuickStep databases are automatically distributed among a group by the QuickStep platform. Similar to native database services, the actual content of records is not of interest for the distribution mechanism and can only be interpreted by the application. Especially, the QuickStep server does not know the record structure.

Conflicts
Concurrent updates on shared data sometimes cause conflicts. Many platforms described above have complex mechanisms to detect and resolve conflicts. In our opinion, such mechanisms cannot be used inside handheld devices. Our concept for solving conflicts is simply to avoid them: it is not possible to concurrently manipulate data. For this, each record of data can only be changed by the handheld device which originally created the record. Copies residing on other handheld devices can only be viewed. To modify data which were created by another user, one has to make a private copy, which is treated as a new record.

Mirroring and Caching
Due to the low computational power of handhelds, heavy processing tasks should run on the server. On the other hand, with respect to the low network bandwidth, it is not possible to transfer a large amount of processing results between server and handheld. To reduce network traffic and to perform as many computation as possible on a server, we developed a combined mirroring and caching mechanism. Each handheld has its *local database* which stores the user's data. A *cache database* stores all data of other users, the local user has currently in view. E.g., the cache database in the calendar tool stores dates of other users in a specific month. Finally, the QuickStep server has a copy of each local database, the *mirror database*. The mirror and cache databases are incrementally updated, each time a handheld device is connected to the server. The application developer has not to worry about the cache and mirror databases; they are completely set up and maintained by the QuickStep runtime system.

5.4 Developing with QuickStep

Figure 4 shows the environment, in which a QuickStep application is embedded.

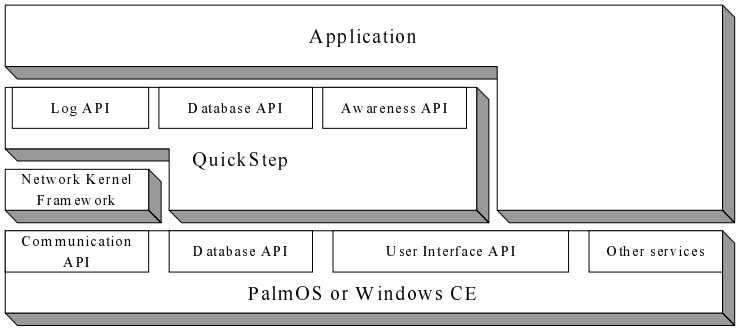

Fig. 4. The QuickStep programming environment

Applications developed with QuickStep use the QuickStep API as well as the API offered by the corresponding operating system. QuickStep is built upon the database communication and user interface APIs. QuickStep does not use the operating system's communication API directly. Instead it uses an intermediate layer, called the *network kernel framework*. This layer which has been introduced by the DreamTeam platform [14] offers a generic interface for communication services such as starting and stopping connections, transferring data etc. With the help of the network kernel framework it is possible to exchange the underlying communication API without changing the QuickStep platform. E.g., we can exchange a TCP/IP communication by a direct serial or Infrared connection and only have to adapt the network kernel framework.

5.5 Privacy

To ensure privacy, QuickStep does not transfer any private data across the network. Every record can be marked as *private* (the default value). Private records reside only on the handheld and will not be transferred in any case.

Non-private records are not transferred until an anonymising process relieves them from personal fields. Since the record structure is opaque to the underlying system, the anonymising function has to be provided by the application. E.g., in the calendar application, the anonymising function blanks out the labels of appointments and transfers the date range only. E.g. the entry

```
May 11-13: "Jörg is on the EHCI"
```

will be transformed to

```
May 11-13: "Jörg is away"
```

since others do not have to know anything about the reason of absence.

As an additional concept, each record has a 'time to live' entry, after which a record is deleted automatically from the QuickStep server and other handheld devices. This is done because a user wants to be sure that her or his data are not available eternally on other computers (even in anonymised form). The time to life entry can be one of *session*, *min*, *hour*, *day* and *forever*. If the value is *session*, the corresponding record will be immediately removed from the server and handheld caches after the corre-

sponding handheld is disconnected. The other values indicate the time, a record will reside after disconnection. The lifetime is controlled by special tasks inside the platform, the *lifetime supervisors* which exist on the handheld devices and on the QuickStep servers.

A similar concept applies to the space property. Each record has a 'space to live' entry, which tells to which servers a corresponding copy is transferred. The space to live entries can be one of *location*, *organisation* and *everywhere*. If the value is *location*, only the QuickStep server which serves the current location gets a copy in the corresponding database. If the value is *organisation*, the record is transferred to all servers inside the organisation. The value *everywhere* is for future use and currently not supported. We work on a concept which allows to link multiple organisations together in order to exchange data. Currently, mirroring and group management requires a tight coupling between the servers, thus is only suitable for local area networks. Transferring data between organisation requires completely different mechanisms.

5.6 Awareness

Context information are both important for users as well as for the application, which may make decisions based on contextual data. A user who collaborates wants to know about the context she or he is currently working in. For this, a user can open a frame as presented in figure 5.

Fig. 5. The context frame

The context frame is the central instance for all context-related information:

- What is the current connection state (connected or disconnected)?
- To which server is the handheld currently connected (server name, organisation)?
- What is my current location?
- Who can be called in case of problems (e.g. network failures)?
- Which other users are currently in the same location or organisation?
- What are their connection states?

The location information are important when a user enters an unknown location. Consider a scenario where a huge building is equipped with a number of QuickStep servers (e.g. one per floor). Each QuickStep server provides information about the current location and thus can be used as a beacon for navigating inside the building.

For collaborating users, the connection state is very important. If a user is disconnected, all changes applied to data cannot be viewed by other users. Thus, information about the connection state should be available on the main window of an application. We designed an integrated button and state indicator (figure 1, lower right button). This widget allows to connect and disconnect to a QuickStep server and indicates the current state with the help of a small icon.

The button/state indicator as well as the context frame are predefined awareness widgets and can be integrated in an application with help of the QuickStep library. In addition, an application can retrieve state and context information via the QuickStep API and can react on events (e.g. disconnecting from a network). With this, an application developer can create his or her own awareness widgets.

5.7 Usability

QuickStep is explicitly designed for handheld scenarios and respects hardware and software limitations. As described above, heavy computation is avoided and network limitations are considered. Using a well-formed and tested platform a developer can rely on stable services. The database abstraction offers a suitable application framework. All services related to communication are embedded inside the platform. To easily find errors in the application itself, a developer can use the log API (see figure 4). Handheld logs are stored on QuickStep servers, thus problem analysis is possible, even when an application or the entire handheld device crashes.

Respect everyday requirements:
Everyday tasks often run unplanned. In order to encourage spontaneous communication between users, central administration has to be avoided. For this, groups of interacting users are not defined explicitly in QuickStep. All users connected to a specific QuickStep server at the same time and using the same QuickStep application form a collaborative session. This concept allows to run a server without defining groups centrally. It is possible for a user to join a group without having explicit permission by existing users. Since a mechanism to anonymise data is integrated into the platform a user cannot spy out private data. QuickStep does not provide services for leaving a collaborative group. When a user disconnects, the server first assumes a temporary disconnection which happens frequently. Only if a user is disconnected for a longer time (e.g. an hour), the server takes that user off the session. The period of time, a user has to be disconnected until a leave operation is performed, is defined by the corresponding application. When a user leaves, the corresponding mirror database is deleted from the server.

6 Conclusion and future work

Handheld applications require fundamental other approaches than desktop applications. If, in addition, applications should exchange information between users, additional issues have to be considered. We identified three properties, an ideal communication-oriented handheld application has to meet: *privacy*, *awareness* and *usability*.

As a realisation which meets these requirements, the QuickStep approach is presented. QuickStep allows to develop mobility and collaboration aware applications and has been especially designed for handheld devices. The generic QuickStep server relieves the handheld devices from heavy tasks and stores data during disconnection. The QuickStep server operates without human intervention and can serve arbitrary QuickStep applications without modification. A server offers contextual information, which can be used by handheld applications. Data distribution is handled by a caching and mirroring mechanism.

In the future, we will follow two directions. First, we more want to include traditional computers into the approach. Currently handheld computing both relies on handheld and desktop applications. Data input is more conveniently on desktop computing, thus an appropriate concept has to support both kinds of computers.

Second, we want to extend QuickStep to a global communication infrastructure. With this, two or more users operating at different places in the world could exchange data. Since wide area connections are considerable slow compared to a local area network, we have to discover new concepts. New technologies such as UMTS may help to address this problem.

References

1. Abowd G. D., Mynatt E. D.: Charting Past, Present and Future Research in Ubiquitous Computing, ACM Transactions on Computer-Human Interaction, Special Issue on HCI in the new Millennium, Vol. 7, No. 1, March 2000, 29-58
2. Bey C., Freeman E., Mulder D., Ostrem J.: Palm OS SDK Reference, 3Com, http://www.palm.com/devzone/index.html, Jan. 2000
3. Boling D.: Programming Windows CE, Microsoft Press, 1998
4. Brors D.: Software Highlights für Palm-Rechner, C'T Vol. 7, Apr. 2000, 138-141
5. C-Technologies, ARIPOS Programming, http://www.cpen.com
6. Demers A. J.: Research Issues in Ubiquitous Computing, Proc. of the thirteenth annual ACM symposium on Principles of distributed computing, Aug. 14-17, 1994, Los Angeles, 2-8
7. Ellis C. A., Gibbs S. J., Rein G. L.: Groupware - some issues and experiences, Communications of the ACM, Vol. 34, No. 1, Jan. 1991, 39-58
8. Joseph A. D., Tauber J. A., Kaashoek M. F.: Mobile Computing with the Rover Toolkit, IEEE Transactions on Computers, Vol. 46, No. 3, March 1997 337-352
9. Kistler J. J., Satyanarayana M.: Disconnected Operation in the Coda File System, ACM Transaction on Computer Systems, Vol. 10, No. 1, Feb. 1992, 3-25
10. Lotus Development Corporation: Lotus Notes, http://www.lotus.com/home.nsf/welcome/lotusnotes
11. Microsoft Outlook, http://www.microsoft.com/outlook
12. Munson J. P., Dewan P.: Sync: A Java Framework for Mobile Collaborative Applications, special issue on Executable Content in Java, IEEE Computer, 1997, 59-66

13. Point-to-point short message service support on mobile radio interface, http://www.etsi.org, Jan. 1993
14. Roth J.: DreamTeam - A Platform for Synchronous Collaborative Applications, AI & Society (2000) Vol. 14, No. 1, Springer London, March 2000, 98-119
15. Roth J., Unger C.: Using handheld devices in synchronous collaborative scenarios, Second International Symposium on Handheld and Ubiquitous Computing 2000 (HUC2K), Bristol (UK), 25.-27. Sept. 2000
16. Stabell-Kulø T., Dillema F., Fallmyr T.: The Open-End Argument for Private Computing, First International Symposium on Handheld and Ubiquitous Computing, Karlsruhe, Germany, Sept. 1999, Springer, 124-136
17. Tasker M., Dixon J., Shackman M., Richardson T., Forrest J.: Professional Symbian Programming: Mobile Solutions on the EPOC Platform, Wrox Press, 2000
18. Terry D. B., Theimer M. M., Petersen K., Demers A. J.: Managing Update Conflict in Bayou, a Weakly Connected Replicated Storage System, Proceedings of the fifteenth ACM symposium on Operating systems principles, Copper Mountain, CO USA, Dec. 3-6, 1995, 172-182
19. Weiser M.: The computer for the Twenty-First Century, Scientific American, 1996, Vol. 265, No. 3, Sept. 1991, 94-104
20. Wireless Application Protocol Architecture Specification, WAP Forum, http://www.wapforum.org/, April 30, 1998