THE RESOURCE FRAMEWORK FOR MOBILE APPLICATIONS
Enabling Collaboration Between Mobile Users

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

Keywords: Mobile Applications, Collaborative Applications, Application Framework, Development Platform

Abstract: Mobile devices are getting more and more interesting for several kinds of field workers such as sales representatives or maintenance engineers. When in the field, mobile users often want to collaborate with other mobile users or with stationary colleagues at home. Most established collaboration concepts are designed for stationary scenarios and often do not sufficiently support mobility. Mobile users are only weakly connected to the communication infrastructure by wireless networks. Small mobile devices like PDAs often do not have sufficient computational power to handle effortful tasks to coordinate and synchronize users. They have for example very limited user interface capabilities and reduced storage capacity. In addition, mobile devices are subject to other usage paradigms like stationary computers and often turned on and off during a session. In this paper, we introduce a framework for mobile collaborative applications based on so-called *resources*. The resource framework leads to a straightforward functional decomposition of the overall application. Our platform *Pocket DreamTeam* provides a runtime infrastructure for application based on resources. We demonstrate the resource concept with the help of two applications built on top of the Pocket DreamTeam platform.

1 INTRODUCTION

Software which enables collaboration between users, so-called *groupware*, allows users to cooperate even when they are geographically distributed. Groupware plays an essential role for shared document editing or cooperative software development. Field workers can use synchronous groupware to discuss shared documents such as manuals or service instructions with their colleagues at home. In this paper, we focus on *synchronous* collaboration, where users at different locations work together *at the same time*. Groupware platforms help to decrease the development costs for a synchronous groupware drastically as they take over a number of tasks of synchronous sessions. As a result, an application developer can concentrate on application-specific details. As development cycles are very short, we can apply rapid prototyping concept and involve the end-user very early in the design process.
A groupware platform usually covers three areas (Roseman & Greenberg, 1996; Dewan & Choudhary 1992):

– an application framework provides a frame for the application development;
– a runtime system offers services such as group, user and session management at runtime;
– a number of interfaces, abstractions and objects allow the developer to use platform services and hide implementation details.

If end-users are mobile, some established concepts of existing platforms are not longer applicable. Such concepts often depend on stationary workstations with high computational power, comfortable user interfaces and reliable, broad-banded networks. In this paper, we introduce an application framework, which was especially designed for mobile users in synchronous sessions. For our approach, we made the following assumptions:

– Synchronous sessions have both mobile as well as stationary participants.
– Mobile participants use mobile devices like PDAs or handhelds as shown in fig. 1. In principle, we could consider notebooks, but their computational power and interface capabilities can be compared to stationary PCs. Therefore notebook computers are not discussed here.
– The network connections of the mobile devices use wireless communication technologies. In our test environment, we use Wireless LAN (IEEE 802.11b) but also mobile phone networks such as GSM or UMTS as well as wireless personal

area networks (e.g. Bluetooth, IrDA) are possible. We assume the existence of a stationary core network.
– The support of stream media (e.g. audio and video) is not object of this work. We assume that there is a corresponding communication channel for voice transmission, e.g. based on a mobile phone network.

Fig. 1: PDA with a Collaborative Application

2 RELATED WORK

Creating groupware platforms for synchronous teamwork has a long tradition. The platforms often are significantly different regarding the means of expression and abstractions for groupware tasks. *Habanero* (Chabert et al., 1998) for example distributes user events, synchronized by a central server. *Groupkit* (Roseman & Greenberg, 1996) uses a central server for the session management only, applications run replicated. The *ALV* model (Hill et al., 1993) allows the developer the express consistency conditions between the data model and the user interface. Many platforms base on multi-user variants of *MVC* (Graham, 1996; Schuckmann et al., 1996). A number of additional models, particularly *PAC**, are based on the *PAC* model (Coutaz, 1997).
All aforementioned models and platforms take an idealized view on networks and involved computer and concentrate on issues of user computer interaction. Mobility of users particularly remains unconsidered. Some newer platforms weaken the idea of strong synchronous cooperation to express mobility issues. *QuickStep* (Roth & Unger, 2001) introduces the idea of *relaxed synchronous collaboration* when mobile participants are loosely coupled to the communication infrastructure and often disconnected from other participants. Further platforms like *Sync* (Munson & Dewan, 1997), *Coda* (Kistler & Satyanarayana, 1992) or *Rover* (Joseph et al., 1997) concentrate on conflict resolution of concurrent data accesses in mobile environments.

3 POCKET DREAMTEAM

To examine the consequences of end-user mobility in synchronous group environments, we extended our groupware platform *DreamTeam* (Roth, 2000) for mobile usage. DreamTeam first was designed for stationary users such as personal of a company, which either work at their office or are connected by modem connections from home. With the platform extension *Pocket DreamTeam,* we now want to support mobile users. Pocket DreamTeam extends the runtime system, the development environment and the application framework. Before we present the mobile extension, we briefly outline the stationary variant of DreamTeam.
DreamTeam is based on a completely decentralized architecture, i.e. apart from the users' workstations or PCs no further computers (e.g. servers) are needed. This architecture is ideal for a huge number of scenarios, where a central server is too cost-intensive or inappropriate for the intended task.
The runtime system of DreamTeam offers several services for the coordination of the participants, group and user profile management, session management and announcement services. Pre-defined elements to achieve group awareness (e.g. participant lists, distributed mouse pointers or overview windows) can be integrated into the application with view lines of code. DreamTeam applications are developed in Java. A class library of approx. 200 Java classes supports the developer.

3.1 The Stationary Application Model

Applications under DreamTeam are developed according to the *DreamTeam Resource Model* (*DRM*). Applications consist of an *application frame*, a set of *resources* and a *user interface*. The application frame links together all other components and provides an interface for the runtime system. With this interface, the runtime system can initialise, start and stop applications. In addition, the system can start

applications in private mode. A user can for example prepare documents for a collaborative session privately.
Resources represent the shared state of an application. Resources can for example be the content of shared web pages, shared paragraphs of text documents or diagram elements of a shared diagram. Resources both provide the data as well as the necessary functions for collaborative processing.

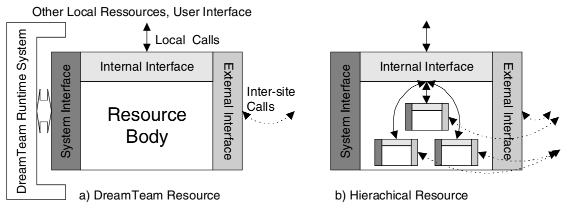


Fig. 2: DreamTeam Resouces

Resources have three interfaces (fig. 2a):

- The *internal interface* is the method interface, which a resource provides by its implementation as a Java class. All application objects can use this interface according to traditional method call mechanisms of Java.
- An *external interface* is used to communicate to corresponding resources on other computers by so-called *inter-site calls*. These calls are method calls which are executed synchronously on all corresponding replicated resources. The developer indicates inter-site calls in the source code by a certain keyword. The runtime system uses the reflection API of Java to invoke replicated method calls.
- The *system interface*: a resource must offer services that make it possible for the runtime system to get control over the resource. The runtime system can transmit the status of a resource to latecomers automatically or maintain consistency during synchronous state changes initiated by different users.

Fig. 3 presents the architecture of DreamTeam applications. This architecture has several advantages. On each site runs a complete set of application instances. All data is available locally, thus an application is still runnable in case of network problems. Inside a local application instance, resources can be used like other object, thus an application developer deals with established paradigms for software development. Not only several applications can be executed simultaneously. It is also possible to open more than one instance of a single application inside a session.

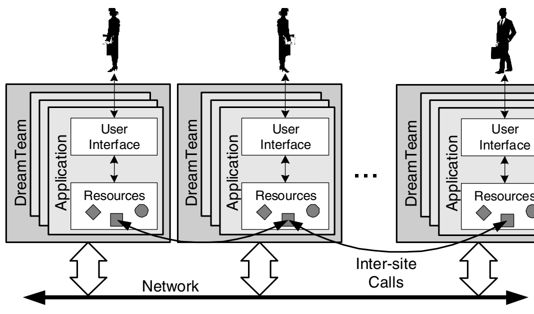


Fig. 3: Stationary Applications with DreamTeam

The external interface connects a resource with its corresponding resources on other peers. Thus, a specific resource and its communication capabilities can be developed without the knowledge of other resource. This leads to a modular software architecture.
Via the system interface the runtime system gets the necessary control over the resources. The developer does not have to take care of the distribution of shared data, the support of latecomers, the synchronization of accesses etc. Particularly the complex area of communication is completely hidden from developer.
The resource model represents a flexible framework with covers different other application models such as MVC, PAC or ALV. Resources are not restricted by their complexity. Resources can be built up by other resources and thus represent small applications inside an application (fig. 2b). Hierarchically built resources form the basis of the component concept *TeamComponents* (Roth & Unger, 2000).

3.2 Mobile Users

The mobility of participants leads to a number of problems, which have far-reaching effects on the runtime environment, on the application framework and on the application development.

- Mobile devices have reduced capabilities regarding the user interface, have a low size and screen resolution and no or only a rudimentary keyboard. Usual interface paradigms of stationary environments cannot be transferred directly to mobile computers. Overlapping windows, icons, drag and drop, context menus etc., are not suitable for devices with small screens. Particularly the concept of direct manipulation is problematic (Kristoffersen & Ljungberg, 1999). To attain the overview on small displays, special dialog widgets and design guidelines were developed

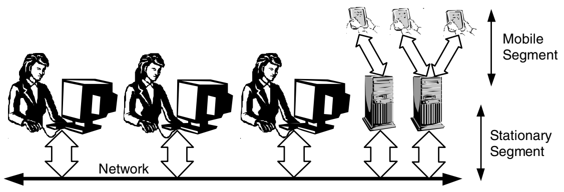


Fig. 4: The Distribution Architecture of Pocket DreamTeam

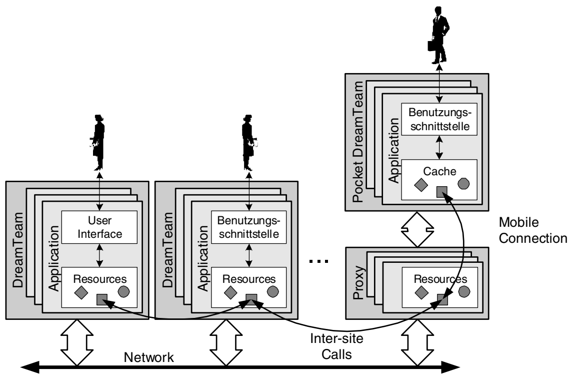


Fig. 5: Applications with Pocket DreamTeam

which are significantly different from those of the traditional computers.
- Usually, mobile devices have a mobile power supply. Battery lifetime still is a limiting factor. If a device is continuously switched on, current batteries often only have power for some hours. The mobile device is therefore turned off most of the time or is in a power saving mode with reduced activity.
- Mobile devices are usually connected wirelessly to a network. Wireless networks have poor characteristics regarding bandwidth, latency time and reliability.
- Finally, technical properties of mobile devices have to be taken into account, which are low processor performances, small memory and no or only rudimentary file systems. Operating systems of mobile devices offer, compared to desktop operating systems, only a low amount of services.

Although mobile devices have limited capabilities, end-users expect very short response times of the applications, even shorter as for desktop applications. Bey et al. (2001) specify a maximum response time of one second for handheld applications. Furthermore, users expect to be able to turn the mobile device off any time and continue the work with the current state later. This usage is significantly different from usage of stationary computers: while a desktop computer is switched on for hours, mobile devices are frequently turned on and off, and sometimes run only for some seconds.
The restrictions on one hand and the high demands on the other hand lead to an architecture as represented in fig. 4.
Besides the mobile devices, we need additional computers called the *proxies*. A proxy executes computational expensive operations of the application. While the mobile device is turned off or the connection interrupted, the state of the session is updated by the proxy. For any stationary session participant, the proxy represents a permanently available contact point. A proxy runs without user invention, thus needs no user interface. With this architecture, arbitrary combinations of mobile and stationary users can cooperate inside a session. From the view of the network, both user types behave identically.
The general idea of a proxy is actually not very new. The first proxy architecture designed for networked applications was introduced by Shapiro (1986). Systems use Shapiro's proxy architecture whenever an application wants to use a specific service, but the actual service location and usage conditions may vary during runtime. Typical examples are CORBA and Jini. With the help of a proxy, a client can use a service without knowing the underlying protocol. However, this proxy resides on the client device, thus does not allow any load balancing between client and other computers.
An example, which is closer to our intended proxy architecture is the HTTP proxy (Fielding et al. 1997). HTTP proxies convey HTTP requests from a web browser to a web server and in turn transfer the requested data back to the client. As a benefit, a proxy can cache web pages, which speeds up access to frequently used pages. However, this kind of proxy only works in one direction, since clients do not offer any services themselves as in our framework. In addition, such proxies are intended to increase the network throughput, not to handle complete disconnections as our proxy. As clients cannot modify a shared state, the entire problem of consistency and coherence is not relevant for such service proxies.
At first sight, proxy computers are a break in the decentralized architecture of DreamTeam. However, there may be an arbitrary number of proxy com-

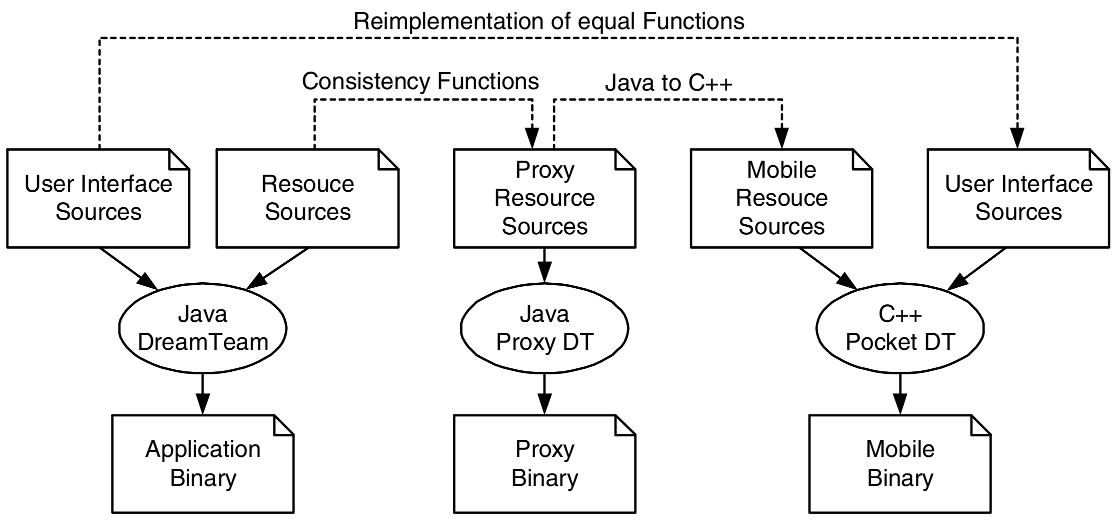


Fig. 6: Steps to Develop Applications

puters in the architecture, thus the failure of a specific proxy computer particularly does not mean that a mobile user is disconnected from the session. Automatic recovery mechanisms switch a user to another proxy without interruption.

By the introduction of proxy computers, the application framework must be modified (fig. 5). A proxy computer maintains the resources for one or more mobile users. Inter-site calls are executed in the proxy. The state therefore remains up-to-date, even if the connection to the mobile computer is interrupted. The mobile computer stores a cache entry of every resource. The mobile application can access the data of the resources locally without executing long network transactions. As a result, the user can continue the work during interruptions for a limited time.

This comfort leads to more complex protocols for data distribution, cache coherence and consistency (Roth, 2002). Particularly the consistency control is crucial. Original DreamTeam uses pessimistic concurrency control procedures, which are suitable for optimally connected users in stationary networks. Pessimistic mechanisms lead to efficiently implementations and are easy to maintain by an application developer. Among mobile, weakly connected users however, pessimistic concurrency control mechanisms are not suitable any longer. As a solution, Pocket DreamTeam offers a combination of a pessimistic concurrency control for the stationary segment and an optimistic concurrency control for the mobile segment. The runtime system of Pocket DreamTeam keeps the problems of the consistency control away from the application developer as far as possible, but the developer has to program special procedures that support the runtime system performing the optimistic concurrency control.

3.3 The Development of Mobile Collaborative Applications

Software development for mobile devices is in principle more cost-intensive than the development for stationary computers. Developing groupware which shall run both on stationary and mobile computers another problem arises: both the operating system and the programming environment is differently. DreamTeam applications are developed under Java, Pocket DreamTeam applications run under C++. Cross-platform approaches (e.g. Java ME) are available in principle, but in reality not acceptable due to a number of technical limitations.

Despite these restrictions, rapid prototyping approaches should still be applicable with Pocket DreamTeam. This goal is accomplished by a strong reuse of source code.

Fig. 6 shows the steps to develop Pocket DreamTeam applications. As a first step, a developer creates a stationary application variant. For this the source codes of user interface as well as the code for resources have to be generated.

In a second step, the developer derives the proxy variants of the resources. For this, the developer must generate code to support the optimistic consistency control.

In the third step, the mobile resources have to be generated. Since the mobile part is developed with another programming language, the syntax of the resources must be translated correspondingly. Usually, not the complete resource has to be ported - only the internal data structures and methods for reading the state are necessary. Inter-site calls are passed on to the proxy by the runtime system automatically, thus no porting costs arise here. Until now, the application developer derives the resource sources manually. This transformation should be done semi-automatically in future with the help of a tool. For this, we have to extend the syntax of the source to make use of a converter possible.

At present, the greatest overhead is arising by generating the user interface for the mobile device. User interfaces can be taken over only regarding content. A direct transfer is not reasonable since any target platform has its own optimised interface toolkit, widgets and design guidelines. Possible solutions may be approaches based on so-called *User Interface Plasticity* (Calvary et al., 2001), where a developer designs an interface once and derives the necessary realization for the target platforms automatically. Such approaches are object of intensive research and at present have not led to any usable tools.

3.4 Communication Issues

The communication between client and proxy plays an important role inside our architecture. We have to consider a number of issues:

– The communication link usually uses wireless technologies. As described in (Bakre & Badrinath, 1995), common transport protocols are optimised for wired links and often have poor performance over wireless links. In addition, wireless links do not offer the same variety of protocols. Actual implementations of IrDA, Bluetooth or GSM support the TCP/IP suite only with some serious drawbacks.
– A mobile device has to look up its corresponding proxy at runtime. The relation between proxy and client may change, when the client moves into another network. There exist a number of platforms to support service discovery in unknown environments such as SLP (Service Location Protocol) (Veizades et al., 1997) or Jini (Sun Microsystems, 2000). Some wireless network stacks provide their own discovery mechanisms, such as IAS (Information Access Service) in IrDA or SDP (Service Discovery Protocol) in Bluetooth.
– Client and proxy devices fundamentally differ regarding internal data representation. To exchange data between both devices, we need a machine- and language-independent encoding and decoding scheme. Data could be encoded using XML or Mime types.

As a communication basis for Pocket DreamTeam which addresses these issues, we use our own communication platform *NKF* (*Network Kernel Framework*) (Roth, 2002b). NKF is a middleware platform for small devices such as PDAs or digital cameras, as well as for traditional desktop systems. It is especially designed for supporting new devices, which do not come along with publicly available middleware platforms such as CORBA or RMI. In such cases, developers have to implement the middleware in addition to the actual application, thus NKF is easy to realize and does not make high demands on target platforms. Though we implemented NKF in Java and C, NKF does not rely on a specific programming language paradigm. NKF is based on network stacks such as IrDA or Bluetooth. Special features of a specific network stack such as service discovery or security functions can be accessed via NKF in a protocol-independent manner. Inside NKF, there exists a lookup and service discovery module, which allows an application to look up services inside the network. If the underlying network stack comes along with its own service discovery mechanism, it is used by NKF. If not, NKF provides an own mechanism.

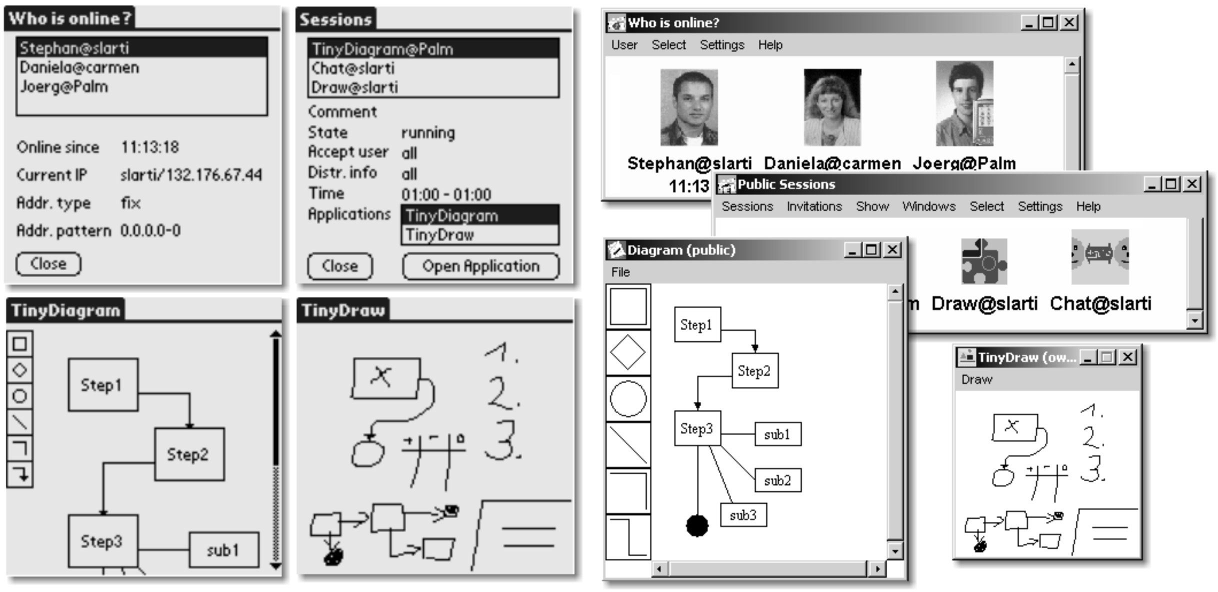
For encoding and decoding data, NKF offers a variety of so-called *codec* modules. NKF contains an XML codec to encode standard data types such as strings, numbers or dates. To transfer complex data such as the session profile, Pocket DreamTeam has to provide an additional marshalling/unmarshalling mechanism. For this, the Pocket DreamTeam development environment contains marshalling interfaces for both Java as well as C++, which are compatible to each other.

3.5 Sample Applications and Evaluations

To verify the concept, two core applications of DreamTeam as well as two cooperative applications were ported for mobile devices with the help of Pocket DreamTeam. It should particularly be verified, how far Pocket DreamTeam supports rapid prototyping.

For implementing the user interfaces, the capabilities of the different platforms were taken into account. While icons, overlapping windows, large menus etc. are common in desktop environments, the mobile variant was reduced to the essential functions. We avoid icons. Some functions of different dialog frames were integrated into a single frame to save time-consuming switches between windows.

The following applications were ported for the mobile platform (fig. 7):


Who is online?
Stephan@slarti
Daniela@carmen
Joerg@Palm
Online since 11:13:18
Current IP slarti/132.176.67.44
Addr. type fix
Addr. pattern 0.0.0.0-0

Sessions
TinyDiagram@Palm
Chat@slarti
Draw@slarti
Comment
State running
Accept user all
Distr. info all
Time 01:00 - 01:00
Applications TinyDiagram
TinyDraw

Fig. 7: Pocket DreamTeam Windows (left) and DreamTeam Windows (right)

– The *online list* shows, which group members are currently active, therefore are potential partners for sessions. This list represents an essential tool for group awareness and allows the users to create spontaneous sessions.
– The user can plan sessions, announce them and join sessions with the help of the *session management tool*.
– With a collaborative *diagram* application, entity relation ship diagrams, class diagrams or flow charts can be developed in the group.
– With a collaborative *free-hand drawing* tool users can create sketches in the group, e.g. for a brainstorming session.

Goal of these implementations was, besides testing the entire system, to assess the costs for application development. An experienced developer needed less than two working weeks for all four applications in the sum. It is problematic to quantify the costs of the developments exactly, since these are very different from developer to developer. We present the numbers of the lines of code here. We know these represent only a trend and cannot be seen as an absolute measure.
Besides the four applications, the overhead of the development of the core platform is shown in table 1. The columns *Stationary*, *Proxy* and *Mobile* show the costs of the program parts of the respective computer category. The column *reused* shows the amount of code of the proxy application, which was reused from the stationary application. We see a very high ratio of at least 90%. In addition, only few lines of code are needed for mobile applications. The reason is that essentially the user interface had to be implemented here and complex functions of the functional core reside in the proxy. The usage of our middleware platform NKF leads to a very lean core platform for mobile devices.

Table 1: Implementation Costs

	Stationary (lines)	Proxy (lines)	re-used	Mobile (lines)
Core Platform	125000	110000	98 %	9500
Online List	4400	4000	95 %	930
Session Management	7100	6900	93 %	2100
Diagram	6900	5200	92 %	600
Draw	930	520	90 %	410

4 CONCLUSION AND FUTURE WORK

Pocket DreamTeam represents a starting point for further researches in the field of the mobile groupware. A developer can create prototypes for mobile collaborative applications economically and include the end-user fast in the development process. This is done by a high amount of reusable source code and a powerful runtime system, which executes demanding services in the background.

Further researches go in two directions. On one hand, the development of a mobile groupware shall further be simplified by the approaches of *User Interface Plasticity*. We expect an enormous potential here.
We will in addition investigate other aspects of mobility. Problem areas arise from the consideration of the spatial position or the current usage context. The complex area of the security in addition plays an important role for mobile users.

REFERENCES

Bakre, A.; Badrinath, R. R.; 1995. I-TCP: Indirect TCP for Mobile Hosts, *15th Internat. Conference on Distributed Computing Systems*, 1995

Bey, C.; Freeman, E.; Hillerson, G.; Ostrem, J.; Rodriguez, R.; Wilson, G.; Dugger, M.; 2001. *Palm OS Programmer's Companion, Volume I*, Palm Inc, July 2001

Calvary, J.; Coutaz, J.; Thevenin, D.; 2001. A Unifying Reference Framework for the Development of Plastic User Interfaces, *8th IFIP Working Conference on Engineering for Human-Computer Interaction (EHCI'01)*, Toronto, May 11-13, 2001, LNCS 2254, Springer, 173-192

Chabert, A.; Grossman, E.; Jackson, L.; Pietrowizc, S.; Seguin, C.; 1998. Java Object-Sharing in Habanero, *Communications of the ACM*, Vol. 41, No. 6, June 1998, 69-76

Coutaz, J.; 1997. PAC-ing the Architecture of Your User Interface, In Proceedings of the DSV-IS'97, 4. *Eurographics Workshop on Design, Specification and Verification of Interactive Systems*, Springer-Verlag, 1997, 15-32

Dewan, P.; Choudhary, R.; 1992. A High-Level and Flexible Framework for Implementing Multiuser Interfaces, *ACM Transactions on Information Systems*, Vol. 10, No. 4, Oct. 1992, 345-380

Fielding, R.; Gettys, J.; Mogul, J.; Frystyk, H.; Berners-Lee, T.; 1997. Hypertext Transfer Protocol - HTTP/1.1, *Request for Comments 2068*, January 1997

Graham, N.; 1996. *The Clock Language: Preliminary Reference Manual*, York University, Canada, 1996

Hill, R. D.; Brinck, T.; Patterson, J. F.; Rohall, S. L.; Wilner, W. T.; 1993. Rendezvous Language, *Communications of the ACM*, Vol. 36, No. 1, Jan. 1993, 62-67

Joseph, A. D.; Tauber, J. A.; Kaashoek, M. F.; 1997. Mobile Computing with the Rover Toolkit, *IEEE Transactions on Computers*, Vol. 46, No. 3, March 1997, 337-352

Kistler, J. J.; Satyanarayana, M.; 1992. Disconnected Operation in the Coda File System, *ACM Transaction on Computer Systems*, Vol. 10, No. 1, Feb. 1992, 3-25

Kristoffersen, S.; Ljungberg, F.; 1999. Designing Interaction Styles for a Mobile Use Context, *First International Symposion on Handheld and Ubiquitous Computing 1999 (HUC'99)*, Karlsruhe, Sept. 27-29, 1999, LNCS 1707, Springer, 281-288

Munson, J. P.; Dewan, P.; 1997. Sync: A Java Framework for Mobile Collaborative Applications, *Special issue on Executable Content in Java, IEEE Computer*, 1997, 59-66

Roseman, M.; Greenberg S.; 1996. Building Real-Time Groupware with GroupKit, a Groupware Toolkit, *ACM Transactions on Computer-Human Interaction*, Vol. 3, No. 1, 1996, 66-106

Roth, J.; 2000. DreamTeam - A Platform for Synchronous Collaborative Applications, *AI & Society* (2000), Vol. 14, No. 1, *Special Issue on Computer-Supported Cooperative Work*, Springer London, March 2000, 98-119

Roth, J.; 2002. Mobility Support for Replicated Real-time Applications (2001), *Innovative Internet Computing Systems (I2CS)*, Kühlungsborn (Germany), June 20-22, 2002, LNCS 2346, Springer-Verlag, 181-192

Roth, J.; 2002b. A Communication Middleware for Mobile and Ad-hoc Scenarios, *International Conference on Internet Computing (IC'02)*, June 24-27 2002, Las Vegas (USA), Vol. I, CSREA Press, 77-84

Roth, J.; Unger, C.; 2000. Developing synchronous collaborative applications with TeamComponents, in Dieng R. et al. (eds): *Fourth International Conference on the Design of Cooperative Systems*, Sophia Antipolis (France), May 23-26, 2000, IOS Press, 353-368

Roth, J.; Unger, C.; 2001. Using handheld devices in synchronous collaborative scenarios, *Personal and Ubiquitous Computing*, Vol. 5, Issue 4, Springer London, Dec. 2001, 243-252

Schuckmann, C.; Kirchner, L.; Schümmer, J.; Haake, J. M.; 1996. Designing Object-Oriented Synchronous Groupware With COAST, In: Proceedings of the *ACM Conference on Computer Supported Cooperative Work*, ACM Press, Nov. 1996, 30-38

Shapriro, M.; 1986. Structure and Encapsulation in Distributed Systems: the Proxy Principle, *Proc. of the 6th Internal. Conference on Distributed Computing Systems*, May 1986, 198-204

Sun Microsystems; 2000. *Jini Technology Core Platform Specification, Version 1.1*, Dec. 2000

Veizades, J.; Guttman, E.; Perkins, C.; Kaplan, S.; 1997. Service Location Protocol, *Request for Comments 2165*, June 1997