An Architecture supporting the development of Collaborative Applications for Mobile Users

Vagner Sacramento, Markus Endler, Hana K. Rubinsztejn,
Luciana dos S. Lima, Kleder Gonçalves, Giulliano A. Bueno
Departamento de Informática, PUC-Rio
R. Marquês de São Vicente 225
22453-900, Rio de Janeiro
{vagner,endler,hana,lslima,kleder,giubueno}@inf.puc-rio.br

Abstract

This article presents a software architecture and services which comprise a middleware infrastructure for the development and operation of collaborative applications for mobile users. The design of the architecture, named Mobile Collaboration Architecture - MoCA*, was driven by the following goals: support scalability in terms of the number of users and services, flexibility and extensibility with respect to the used communication protocols and the application requirements, and facilitate the monitoring, processing and use of context information regarding single users and the collaborative group within the applications.*

Keywords:
Mobile Computing, Middleware, Mobile Collaboration

1. Introduction

As portable computing devices with wireless communication interfaces, such as PDAs with GPRS or IEEE 802.11, Smart-phones, etc become more powerful and common place, the demand for the development of application and services supporting communication and collaboration among mobile users also increases. Although this new distributed computing environment brings new challenges, such as mobility, limited resources on the devices and intermittent connectivity, it also opens a new range of different and yet unexplored forms of user interactions, in which for example information about user locality and proximity plays a distinguished role in determining the form and the participants of an interaction

We argue that collaboration in a static and a mobile network are quite different. While in collaboration environments for static networks, one implicitly assumes an "always-on" connectivity of all user devices, this assumption cannot be made in a mobile setting. Due to the weak and intermittent connectivity in these networks, a user may become temporarily unavailable even though she is still engaged in the collaboration session. Hence, synchronization of views and mutual perception of the collaborating users (i.e. *Collaboration Awareness*) must be redefined in this new context.

Another difference is related to user mobility. When the users are mobile, the group of collaborators tends to be more dynamic, is formed spontaneously, and is motivated by a common interest or situation shared among the peers. Hence, instead of supporting static (and task-oriented) groups like in traditional groupware systems, environments for mobile collaboration should focus on supporting *mobile communities*. Moreover, the form of interaction of a (mobile) user with other community/group members tends to be more variable, asynchronous and dependent on her current context, her activity or current interest. For example, the interaction of a person with a community may be more or less active/engaged depending on her location (e.g. at the working place, or at home), on her activity (e.g. driving, walking, sitting) or even on her state of mind and interest to communicate.

Finally and as already mentioned, collaboration between mobile users is usually not driven by a global and predefined goal or task, such as the cooperative work on a digital or physical artifact, but instead by spontaneous and occasional initiatives to share with others some information, contribute to the development or improvement of a public knowledge. This makes participation in a collaboration be more spontaneous, irregular, and moreover, motivated by implicitly (or explicitly) gain of reputation due to the contribution with higher-quality, more reliable, or more relevant information [8].

All the aforementioned characteristics suggest that envi-

ronments for developing mobile collaboration applications and services should incorporate new mechanisms facilitating the collection, the aggregation and the application-level access to different kinds of information about the individual and collective context of a user or community, which can be both made available to the collaborating peers (e.g. mobile collaboration awareness), or used for adapting the behavior of the application (e.g. available functions or user interfaces) to the current situation.

This paper describes a middleware architecture for developing support services and applications for mobile collaboration named MoCA (*MObile Collaboration Architecture*). The work is part of a wider project which aims at experimenting with new forms of mobile collaboration and implementing a flexible and extensible service-based environment for the development of collaborative applications for infrastructured mobile networks.

In the following session we present a general overview of MoCA, its main components and their interactions. Section 3 presents a simple Chat application with connectivity awareness, which we implemented as a first proof of concept of MoCA. In section 4 we discuss some related work and make a comparison with the MoCA architecture. Finally, in section 5, we make some considerations with regard to the MoCA properties, and mention other ongoing work that in the scope of this project.

2. Overview of MoCA

The *Mobile Collaboration Architecture (MoCA)* was designed for infrastructured wireless networks. The current prototype of this architecture works with an 802.11 wireless network based on the IP protocol stack, but the architecture could as well be implemented for a cellular data network protocol, such as GPRS.

The MoCA infrastructure consists of client and server APIs, basic services supporting collaborative applications and a framework for implementing application proxies (*ProxyFramework*), which can be customized to the specific needs of the collaborative application and which facilitates the access to the basic services by the applications. The APIs and the basic services have been designed to be generic and flexible, so as to be useful for different types of collaborative applications, e.g. synchronous or asynchronous interaction, message-oriented or artifact-sharing-oriented.

In MoCA, each application has three parts: a server, a proxy and a client, where the two first execute on a static node in the wired network, while the client runs on a mobile device. One or more proxies of the application are the intermediates of any communication between the server and the client components.

Applications with requirements to scale to large numbers of clients may have several proxies executing on different networks and interacting with a small(er) number of servers. Actually, the number of proxies required will depend on the type of application, the number of clients and the current demand for interaction by the user community. The proxy of an application may execute several tasks, such as adaptation of the transferred data, e.g. compression, protocol conversion, encryption, user authentication, context processing, distribution of context information, service registration and location, handover management and others. Most of such tasks require quite a lot of processing effort, and hence, the proxy also serves as a means of distributing the application-specific processing among the server and its proxies.

The main services offered by the architecture for the development of collaborative applications are the following:

- *Monitor*: is a daemon executing on each mobile device, which is responsible for (i) collecting state information of the device, such as connectivity quality, energy, CPU usage, free memory, current Access Point (AP), list of all APs and their signal strengths that are within the range of the mobile host, etc, and (ii) sending this data to the CIS (*Context Information Service*), executing on one (or more) node(s) of the wired network.

- *Configuration Service (CS)*: this service is in charge of storing and managing the configuration information for all mobile devices, so that these can use the MoCA infrastructure. The configuration information is stored in a persistent database, where each entry holds the following data: MAC address of the device, the (IP:port) address of the CIS and *Discovery Service*, and the periodicity in which the *Monitor* will send the device's state information to the CIS.

- *Context Information Service (CIS)*: This service receives and processes state information sent by the *Monitor*s. It also receives requests for notifications (aka subscriptions) from application Proxies, and generates and delivers events to a proxy whenever a change in a device's state is of interest to this proxy.

- *Discovery Service (DS)*: is in charge of storing, managing and locating information regarding any application (servers and proxies) registered with the MoCA middleware.

- *APIs* and the *ProxyFramework*: The server and the client of a collaborative application should be implemented using the MoCA APIs. The Proxy of the application will be an instance of the *ProxyFramework*,

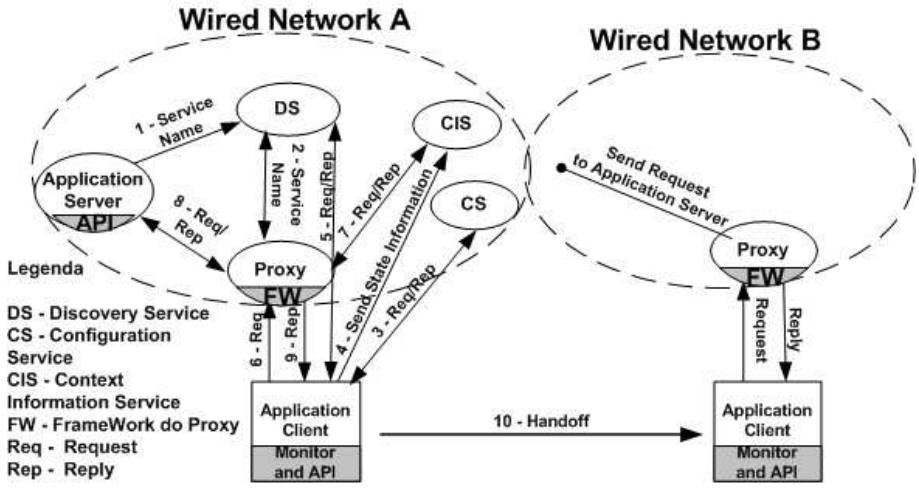

Figure 1. Interaction between basic services and an collaborative application in MoCA

where its behavior is customized according to the specific needs of the application. Both the APIs and the *ProxyFramework* hide from the application developer all the details concerning the use of the services provided by the architecture.

Figure 1 shows the typical sequence of interactions among the elements of the architecture, which is to illustrate the roles played by these elements during registration and execution of an collaborative application, composed of one (or more) instances of an *Application Server*, a *Proxy(ies)* and an *Application Clients*.

Initially, the Application Server registers itself at the DS (step 1) informing the name and the properties of the collaborative service that it implements. Each Proxy of the application also performs a similar registration at the DS (step 2). This way, the Application Clients can query the DS in order to discover how to access a given collaborative service in their current network , i.e. either through the Application Server or a Proxy. Each mobile device executes the MoCA *Monitor*, which polls the current resource and connectivity states of the device, and sends this information to the CIS. The address of the target CIS and the periodicity in which the state is to be sent are obtained from the CS when the device is started (step 3). Thereafter, the Monitor sends periodically the state information to the CIS (step 4).

After discovering a Proxy which implements the desired collaborative service through the DS (in step 5), the client can start sending requests to the Application Server. Every such request gets routed through the corresponding Proxy (step 6), which processes the client's request with respect to specific adaptation needs of the application, and forwards it to the Application Server. For example, the Proxy may send a request to the CIS (step 7) for registering its interest in notifications of some (types of) events concerning the Client it is representing, such as the one given by the following *Interest Expression*, {"FreeMem $<$ 15%" OR "roaming=True"}.

Now, whenever the CIS receives a device's state information (from the corresponding Monitor), it checks whether this state change evaluates any *Interest Expression* to true. In this case, CIS generates a notification event and sends it to all Proxies which have registered interest in such change of the device's state.

When the Application Server receives the client's request (step 8), the request is processed and a reply is sent to some (or all the) Proxies, which will then process (e.g. modify, filter, etc.) the reply according to the context information received from the CIS about the mobile device. Such context-specific processing depends on the specific requirements of the collaborative application. For example, if the Proxy is informed that the quality of the wireless connectivity of a mobile device has fallen below a certain threshold, it could temporarily store the server's reply data in a local buffer for an optimized/bulk transfer, remove part of the data, e.g. figures, apply some compression to the data, etc. Moreover, the Proxy could use other context information, such as the device's location, to determine what data, when and how it should be sent to the client at the mobile device (step 9).

The architecture also implements mobility transparency for the applications. When a mobile devices moves to a new network (step 10), the Proxy performs the *handover* at the application level, e.g. determining the most appropriate Proxy for the device in the new network, and if available, transferring the collaboration session state to this new Proxy. Figure 1 shows a scenario in which the client in network B remains using the proxy of the previous network, (which in turn uses CIS and DS of network A), since supposedly these services are not available in its new local net-

work. But also in case of a successful handover to a new local Proxy neither the Client nor the Server would be aware of the migration. In any case, all of the client's request received by the current Proxy would be transparently forwarded to the corresponding application server.

3. Wireless Chat: a case study

As a first proof of concept of MoCA we implemented a Chat tool which we called *W-Chat* (from Wireless Chat) and which has as its unique feature the diffusion of connectivity status of each participant in a chat room (called *Forum*) and support "catch-up" after a disconnection. To implement this extra feature, W-Chat uses connectivity information provided by CIS.

The W-Chat proxy intercepts all messages (commands and events) from the client to the server, and vice-versa, and has a local message buffer holding the last N undelivered chat messages for every of its clients. It also registers at CIS its interest in any disconnection and connection event from any of its devices. This is done through the following expression, where *MacAddr* stands for the specific MAC Address of a device:

`DeviceID`: *MacAddr*,
`Context`: { (last_state = connected AND curr_state = disconnected) OR (last_state = disconnected AND curr_state = connected) }

The proxy also registers itself at CIS to receive events about new clients requesting its services.

Like most traditional chat services, W-Chat enables users to create new Forums (chat rooms), identify the users currently participating in a Forum, search for Forums by subject, participate simultaneously in several Forums, etc. In addition, W-Chat displays the connectivity status of each Forum participant. Therefore, when a participant becomes disconnected, for example, because it moved to a region without wireless coverage, a characteristic icon showing this new status appears close to the user's name in the list of Forum participants. This additional information about mutual availability for collaboration (i.e. a new form of *Collaboration Awareness*) helps users to decide if they should or not expect immediate messages from other users. Moreover, being (temporarily) disconnected is obviously a different state than leaving a Forum, and hence these two states should be perceived differently by the other participants.

When a user reconnects to the network, W-Chat's Client and Proxy synchronize their states, and the user gets the N most recent messages of each of the Forums in which she was participating. For the communication between the W-Chat client and Proxy we used JMS[1].

The MoCA architecture supported the implementation of W-Chat application by handling transparently all problems related to the mobility and temporary disconnection of the clients and by offering a high-level interface to a service for obtaining context information about the mobile devices, e.g. the wireless connectivity.

Figure 2 shows a screen-shot of W-Chat's client for Windows on notebooks[1], where the right window shows an open Forum (named MoCA), with a list of 5 participating users, of which two of them are currently disconnected from the (wireless) network, as indicated by the crossed icons. The left side window displays the set of all Forums available to a user at a given moment. This window also allows her to receive any message not related to a specific forum.

4. Related work

There are several other works which aim at providing support for some sort of collaboration in mobile networks. In this section we present and discuss some work most closely related to our proposed architecture. Initially we emphasize main characteristics of each system or environment, and at the end of the section we compare them with MoCA.

YACO (Yet Another Collaboration Environment) [4] is a *framework* for collaborative work, which is based on SIENA [5], a distributed, content-based Pub/Sub communication infrastructure, and on MobiKit [3], which is an mobility service toolkit based on proxies. Using MobiKit's operation *moveOut* a client can inform a server of its disconnection. Whenever the client is reconnected to the network it may invoke operation *moveIn*, by which it is able replay all the events missed during the period of time it was disconnected. As a collaboration environment, YACO offers a message service, a service for discovery of users, and a service for sharing of artifacts (files and programs).

The architecture MOTION [6] offers collaboration services such as search and exchange of distributed artifacts (on mobile devices) in a *peer-to-peer* architecture, and a message system based on events *publish/subscribe*. MOTION provides TSW (*TeamWork Services*) for managing user groups and access rights (through its DUMAS subsystem), storage and sharing of artifacts, and support for different mobile devices.

STEAM [7] is an event-based middleware for collaborative applications where location plays a central role. It is a system specially designed for ad-hoc mobile networks, and hence inherently distributed. It supports filtering of event notifications both based on subject and on proximity. This second kind of filtering is used to restrict the propagation of events. Subject- and proximity-based filters can be associated with a event producer, while contents-based filters can be applied by event consumers.

YCab [2] is also a framework for development of collaborative services for ad-hoc networks. It has a flexible

[1] Currently, we are also developing a W-Chat client for Compaq iPAQs

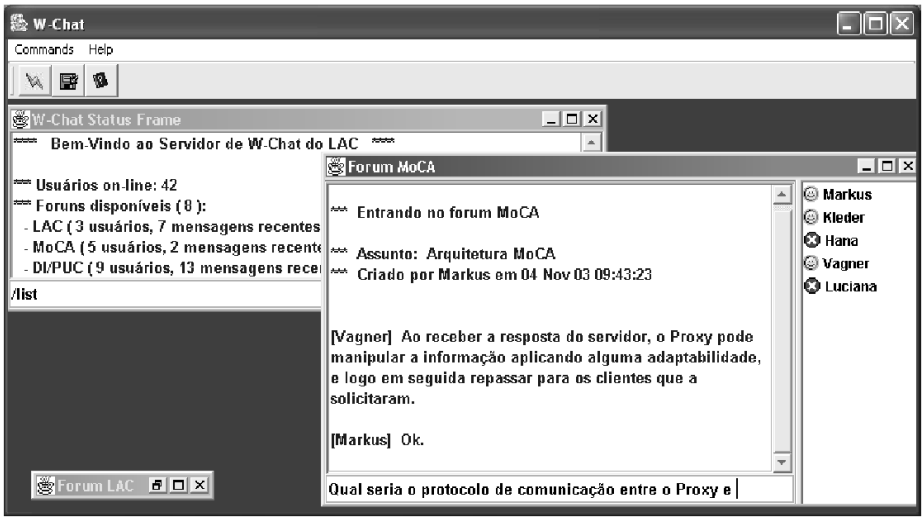

Figure 2. W-Chat Client Windows

API for the development of collaborative applications for mobile users. The framework supports asynchronous and multicast communication based on 802.11. The architecture includes a module for message routing and modules managing the communication, the client component and its state. Moreover, it provides support for decentralized control using algorithms for distributed coordinator election and session management. Among the offered collaboration services, there is a chat, a shared white-board, and sharing of images (video-conferencing) and user files.

Analysing the aforementioned environments, we can see that most of them essentially use the Pub/Sub communication paradigm, which seems to be best suited for environments with weak connectivity between clients and servers, and peers. MoCA also supports event-based communication through appropriate operations at its APIs, but in addition, the application developer can also use others, e.g. connection-oriented (RMI, TCP) or connectionless mechanisms.

Another common feature observed in most systems is their concern to shield from the application developer all aspects regarding mobility and user location, aiming the provision of a seamless, anywhere-available service. Except for STEAM, which handles location information (obtained via GPS receivers), no other work uses information about the current context for triggering appropriate adaptations of the application's behavior (to conform to the new execution environment) or enabling context-specific application functions, such as the proximity-based selection of collaboration partners, or the dissemination of the connectivity status of mobile devices. YACO's *moveIn* and *moveOut* in principle could be used to provide connectivity and approximate location information, but this is not offered by the corresponding API, since YACO's main objective is to hide and not to expose connectivity/location information.

In MoCA, we take a radically different approach, exposing context information for the development of context-aware applications. Through basic services and via the *ProxyFramework* we make available to the application developer a wide range of context information, e.g. the user device's (approximate) location, the quality of the connectivity, the device characteristics, available resources, user preferences, etc., which she can use according to the specific needs of the application.

5. Conclusions

This work is part of a wider project which aims at investigating collaboration support for mobile users. We believe that collaboration among mobile users requires new and different middleware services and functionality than the ones provided by groupware for wired networks.

In particular, we believe that not only individual context information of a user (such as her location or connectivity), but also collective context information (such as the proximity of two or more users) can be used not only to enrich collaboration awareness, but as well allow for new forms of collaboration, which have not been yet explored in conventional, wired collaboration.

Compared with other middlewares and environments for mobile collaboration, MoCA offers a generic and extensible infrastructure for the development both of new services for context services (i.e. for collecting and/or processing context information) and of collaborative applications that make use of this information to determine the form, contents and/or the group of collaboration.

So far, we have implemented the *Monitor*, the *Configuraion Service*, the *Context Information Service* and the *Discovery Service*. We have versions of the Monitor for both Linux and WindowsXP platforms, and they are mostly independent of the 802.11 PCMCIA card being used. For efficiency reasons we have used *sockets/UDP* for the communication between the Monitor and CIS, but between the MoCA basic services and the W-Chat Server and its Proxy we have implemented asynchronous, event-based communication. Concerning the *ProxyFramework* till now we have only a bare-bones implementation, which includes simplified versions of the components *Discovery*, *Caching Management* and *Context Management*, since only these were required for the W-Chat Proxy.

During the development of W-Chat (which took only 2 weeks), we could perceive the benefits of using MoCA's services, APIs and *ProxyFramework*, which reduced considerably the complexity of the application. The current version of W-Chat was implemented using J2SE, but we are now also working on a client for limited devices, using J2ME/CDLC.

In addition, we are also developing a version of the *Monitor* for the platforms Windows, PocketPC e PalmOS. We have also started exploring other alternatives of using context information for creating new forms of collaboration. Using a probabilistic method we are implementing an *Approximate Positioning Service* which is based on the strength of the 802.11 RF signal received from various access points. We are also designing a *Proximity Service* which will use the *APS* and will allow the user to configure her own set of *neighborhoods* with different levels of granularity, and detect the set of other users that are currently co-located in her current *neighborhood*. Using such information we are planning to design and implement new collaboration applications which are proximity-sensitive, e.g. a user entering a (conference) room is automatically included in a chat Forum of that room so that she can also participate in message exchanges among the members of the audience, or detect if there is someone to share files with.

In yet another thread of research, we are investigating means of defining user interests using ontologies, and designing services for the evaluation of affinity and discovery of similar (or complementary) interests. The goal is to design collaborative applications which use both information about co-localization and interest affinity in order to determine the peers and the form of collaboration.

References

[1] *The Java Message Service Specification*. http://java.sun.com/products/jms/docs.html (Last visited: November 2003).

[2] D. Buszko, W.-H. Lee, and A. Helal. Decentralized ad hoc groupware API and framework for mobile collaboration. In *Proceedings of the 2001 International ACM SIGGROUP Conference on Supporting Group Work, Boulder, USA*, Oct. 2001.

[3] M. Caporuscio, A. Carzaniga, and A. L. Wolf. Design and evaluation of a support service for mobile, wireless publish/subscribe applications. *IEEE Transaction of Software Engineering*, 29(12):1059–1071, December 2003.

[4] M. Caporuscio and P. Inverardi. Yet another framework for suporting mobile and collaborative work. In *Proc. of the International Workshop on Distributed & Mobile Collaboration (DMC), at the 12th Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE), Linz, Austria*, June 2003. http://citeseer.nj.nec.com/583641.html (Last visited: August 2003).

[5] A. Carzaniga, D. Rosenblum, and A. L. Wolf. Achieving scalability and expressiveness in an internet-scale event notification service. In *Nineteenth Annual ACM Symposium on Principles of Distributed Computing*, pages 219–227, July 2000.

[6] E. Kirda, P. Fenkam, G. Reif, and H. Gall. A service architecture for mobile teamwork. In *Proc. of the 14th International Conference on Software Engineering and Knowledge Engineering, Ischia, Italy*, 2002.

[7] R. Meier and V. Cahil. Exploiting proximity in event-based middleware for collaborative mobile applications. In *4th IFIP International Conference on Distributed Applications and Interoperable Systems (DAIS'03), Paris, France*, 2003.

[8] H. Rheingold. *Smart Mobs: The Next Social Revolution*. Perseus Publishing, Oct. 2002. ISBN: 0738206083.