MoCA: A Middleware for Developing Collaborative Applications for Mobile Users

Vagner J. do Sacramento Rodrigues, Markus Endler, Hana K. Rubinsztejn,
Luciana Lima, Kleder Miranda Gonçalves, Fernando Ney Nascimento, Giulliano A. Bueno
e-mail: {vagner,endler,hana,kleder,ney,giubueno}@inf.puc-rio.br

PUC-RioInf.MCC44/04 December, 2004

Abstract

This article presents a middleware architecture for the development and deployment of context-aware collaborative applications for mobile users. The architecture, named *Mobile Collaboration Architecture - MoCA* comprises client and server APIs, a set of core services for registering applications, monitoring and inferring the execution context of mobile devices, and an object-oriented framework for instantiating and customizing server proxies according to the specific adaptation and context-processing requirements of the applications. MoCA facilitates the development of distributed programs which require access to individual and group context in order to adapt its behavior. The design focused on simplicity, extensibility, scalability, protocol heterogeneity and application customization.

Keywords: Mobile Computing, Middleware, Context-awareness, Mobile Collaboration.

Resumo

Este artigo apresenta uma arquitetura de middleware para o desenvolvimento e implantação de aplicações colaborativas com percepção de contexto para usuários móveis. A arquitetura, denomindada *Mobile Collaboration Architecture* - MoCA é formada por APIs do cliente e do servidor, um conjunto de serviços para registrar aplicações, monitorar e inferir o contexto de execução dos dispositivos móveis e um framework orientado a objeto para instanciar e customizar servidores proxies de acordo com os requisitos de processamento de contexto e adptações específicas da aplicação. MoCAfacilita o desenvolvimento de programas distribuídos que requerem acessar o contexto do grupo e de um individuo especifico para adptar seu comportamento. O projeto focou na simplicidade, extensibilidade escalabilidade, heterogeneidade de protocolos e na customização da aplicação.

Palavras Chave: Computação Móvel, Middleware, Percepção de Contexto, Colaboração Móvel

MoCA: A Middleware for Developing Collaborative Applications for Mobile Users

I. 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 applications 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-on̈connectivity 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, and it is formed spontaneously motivated by a common interest or situation shared among the peers. 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.

Finally and as already mentioned, collaboration between mobile users is usually not driven by a global and pre-defined goal or task, such as the cooperative work on a digital or physical artifact, but instead by spontaneous and occasional initiatives to share some others' information, contribute to the development or improvement of a public knowledge, etc. This makes participation in a collaboration more spontaneous, irregular, and motivated by implicitly (or explicitly) assessed gain of reputation due to the contribution with higher-quality, more reliable, or more relevant information [13].

All the aforementioned characteristics suggest that environments 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. its behavior, available functions or user interfaces) to the current situation.

This paper describes a middleware architecture for developing context-processing services and context-sensitive 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 infra-structured mobile networks.

In the remainder of the paper we present the main components of the architecture and their interactions (section II), MoCA's support for application programming - through its APIs and the *ProxyFramework* (section III), our current work on context-aware collaborative applications (section IV), related work (section V) and concluding remarks (section VI).

II. Overview of MoCA

The *Mobile Collaboration Architecture (MoCA)* was designed for infra-structured 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.

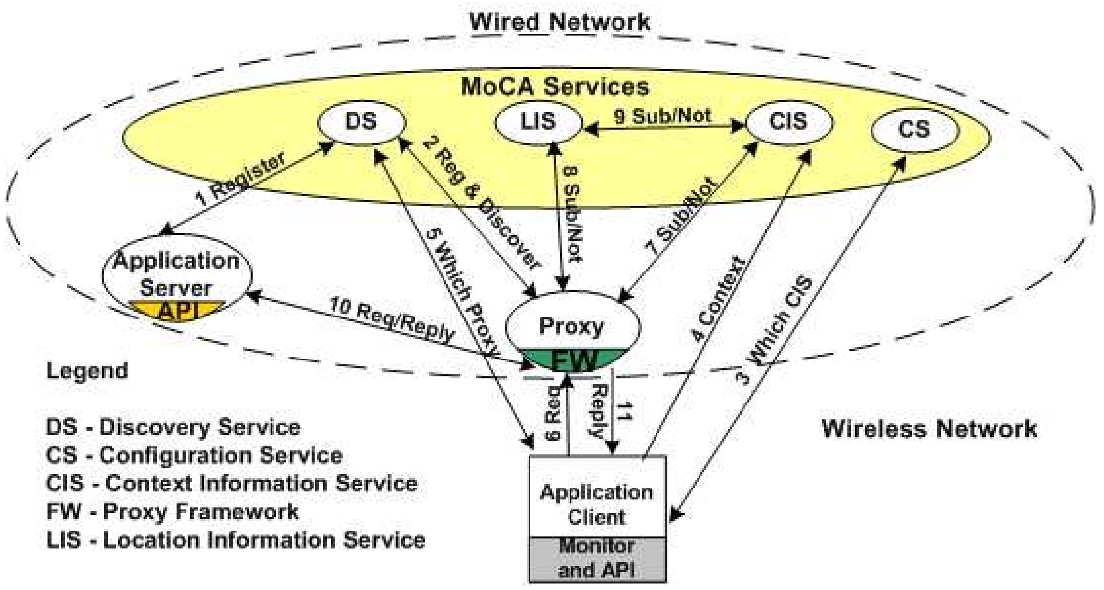

Fig. 1. Typical Interaction Sequence between a collaborative application and MoCA's core services

In MoCA, each application has three parts: a server, a proxy and a client, where the two first execute on nodes of the wired network, while the client runs on a mobile device. A proxy intermediates all communication between the application server and one or more of its clients on mobile hosts.

Applications with requirements to scale to large numbers of clients may have several proxies executing on different networks. The proxy of an application may be in charge of several tasks, such as adaptation of the transferred data, e.g. compression, protocol conversion, encryption, user authentication, context processing, 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.

MoCA offers client and server APIs and a *ProxyFramework*. The server and the client of a collaborative application should be implemented using the MoCA APIs, since they hide from the application developer most of the details concerning the use of the services provided by the architecture (see below). The *ProxyFramework* is a white-box framework for developing and customizing the proxies according to the specific needs of the application. It facilitates the programming of adaptations that should be triggered context-change events.

In addition, the architecture offers the following core services which support the development of context-aware collaborative applications:

- *Monitor*: is a daemon executing on each mobile device and is in charge of collecting data concerning the device's execution state/environment, and sending this data to the CIS (*Context Information Service*) executing on one (or more) node(s) of the wired network. The collected data includes the quality of the wireless connection, remaining 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 device.
- *Configuration Service (CS)*: this service is in charge of storing and managing configuration information for all mobile devices, so that these can use MoCA's core services, such as CIS and *Discovery Service (DS)*. The configuration information is stored in a persistent *hash table*, where each entry (indexed by the device's MAC address) holds the following data: the (IP:port) addresses of a CIS server and a *Discovery Server*, and the periodicity by which the *Monitor* must send the device's information to the CIS. The MAC address-specific indexing is essential for implementing a distributed CIS, where each server gets approximately the same context processing load.
- *Discovery Service (DS)*: is in charge of storing information, such as name, properties, addresses, etc., of any application (i.e. its servers and proxies) or any service registered with the MoCA middleware.
- *Context Information Service (CIS)*: This is a distributed service where each CIS server receives and processes devices' state information sent by the corresponding *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.
- *Location Inference Service (LIS)*: infers the approximate *logical* location of a device. It does this by comparing

the device's current pattern of RF signals received (from all "audible" 802.11 Access Points) with the signal patterns previously measured at pre-defined *Reference Points* in a Building or Campus, using a similar technique as described in [1]. For this, LIS uses the context information of any device collected by *CIS*, and measures the mean value (10 probes) of the *Eucledian Distance* to each Reference point. Since the RF signal is subject to much variation and interference, the location inference is only approximate: its precision depends on the number of access points and the number of the Reference Points. LIS allows the administrator to define logical regions of arbitrary size and shape, and a hierarchical description of regions and its nested sub-regions.

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 *Application Servers*, *Proxy(ies)* and *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. The *Monitor* executing on each mobile device, polls the state of the local resources and the RF signals, and sends this context information to the CIS. As mentioned, the address of the target CIS and the periodicity for sending the context information are obtained from the CS when the Monitor 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 an *Interest Expression* to the CIS (step 7) registering its interest in notifications of events about a state change of the client it is representing. An *Interest Expression* may be for example {"FreeMem $<$ 15%" OR "roaming=True"}.

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

Applications which require location information, instead register their interest with LIS, (step 8) which in turn subscribes at CIS (step 9) for receiving periodic updates of the device's RF signals, which LIS uses to infer the device's location and send the corresponding notification to the Application Proxy.

When the Application Server receives the client's request, the request is processed and a reply is sent to some or all the Proxies (step 10), which may then modify/process the reply (e.g. compress, filter, etc.) according to the notification received from CIS about the corresponding 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 11).

The architecture also implements mobility transparency for the applications. When a mobile devices moves to another network , the Monitor detects this and the CIS notifies the Proxy. The Proxy performs the *handover* at the application level, by determining the most appropriate Proxy for the device in the new network, and if available, transferring the collaboration session state to this new Proxy.

III. Programming Support

In this section we describe the server and client APIs and the *ProxyFramework*, which are to be used when implementing an application using MoCA.

Server API: The server API offers some interfaces which handle the configuration and communication options of the server. Through interface `init(serviceName, properties, key, protType)` the application server registers its name, address and properties at the DS, so that clients can discover it. The communication between the server and its proxies is set by `protType`, and can be either through an event service (using interfaces `subscribe/publish`) or through sockets (using interfaces `send/recv`.

Client API: This API offers interfaces to configure the client, find a proxy, send and receive data from the proxy, and obtain information on its current execution context.

Interface `init(serviceName, properties, protType)` configures the application client so that it can use the services offered by MoCA. It also starts the Monitor as a *daemon*, which collects and sends context data to the CIS. Moreover, it queries DS to discover all the available proxies for the application. This is done by sending the application/service name, its properties and the specific protocol to be used when interacting with the proxy (e.g. SMS, UDP, TCP, JMS, or WAP). With this information, it is possible to select the corresponding implementation of the interfaces (`send`/`recv`, `publish`/`subscribe`) of the client API. Through an interface named `getStatus()` the client can query the context information being collected by the Monitor, which may be used for triggering a local adaptation at the client. This interface returns a reference to a XML-based description of the context information.

ProxyFramework: MoCA offers an object-oriented framework which support the development of the Application Proxy. This framework provides the application developer with simple mechanisms to access context information related to client devices interacting with the server through the proxy, and to define how the proxy should adapt to changes in the client's context.

The *ProxyFramework* offers such facilities through common features and design patterns which are recurrent in most distributed applications using a Proxy as the means to cope with device mobility and intermittent connectivity. The framework consists of interfaces to a set of concrete components (*frozen-spots*), which implement common and pre-defined functionalities of the proxy, and interfaces of a set of abstract components (*hot-spots*), which can be implemented and extended according to the specific demands of each application.

IV. Context-aware Mobile Collaboration

In this section we present two context-aware Collaborative Applications which we are developing as case studies of MoCA. While the first simply uses the connectivity information of the device's context, the second one uses inferred context provided by the LIS service.

W-Chat: W-Chat (from "Wireless Chat") is a simple chat program which has as its distinguished features the diffusion of connectivity status of each participant in a chat room (called *Forum*) and support for "catch-up"after a disconnection, i.e. the client gets all the messages that where posted during the disconnection time. For this, W-Chat uses connectivity information provided by CIS.

The proxy of W-Chat intercepts all messages (commands and events) from the client to the server, and vice-versa, and has a local message buffer for every of its clients, such that 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. The proxy also registers at CIS its interest in any disconnection and connection event from any of its devices.

Since W-Chat displays the connectivity status of each participant, when any of them becomes disconnected, for example, because she 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 chat room 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. W-Chat was implemented in J2ME.

NITA: NITA (from "Notes in the Air") is an application to post text messages (and files, in general) to a logical region as if it were a chat room. Hence, any client which is currently in (or enters) this region and has the proper authorization will automatically receive this message. In the literature we can find several other projects with similar services combining messaging with spatial events [12], [7]. However, most of these services where implemented from scratch or without a general middleware support for context monitoring and inference.

In NITA, the sender of a message can set its destination (a logical region), the users authorized to read the message, and the time period the messages is to be readable. Moreover, it can search for available NITA servers, their regions, and visible users in each of the regions. A potential receiver, can set her visibility flag (on/off), choose which types of messages she wants to receive, and choose between an immediate display of the message, or if it should be logged for future reading.

Because NITA is essentially a message retrieval service driven by spatial events of the kind (*Device X detected in region Y*), and by communication events, it interacts closely with the location service LIS (cf. section II), which provides derived/inferred context information.

The NITA proxy is in charge of querying the LIS service about its Region structure and registering interest in location changes of the clients it represents. Moreover, it manages the client's profile, i.e. whether it should filter out some messages, log the messages or forward them to the client. Although many of these tasks could as well be performed by the NITA server, this decentralization is necessary for providing a scalable service. We have already completed its design, and are now implementing it in J2ME.

V. Related Work

Much research related to middleware and programming environments for mobile and context-aware applications [6], [10] has been done, and many influenced our work. However, due to space limitations, we will only discuss some architectures/environments with similar goals to ours.

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 [9] 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 sub-system), storage and sharing of artifacts, and support for different mobile devices.

ActiveCampus [8] is a large project at UCSD which provides an infra-structure focusing on integration of location-based services for academic communities. It employs a centralized and extensible architecture with five layers (Data, Entity Modeling, Situation Modeling, Environment Proxy and Device) which supports a clear separation of the collection, the interpretation, the association with physical entities and the service-specific representation of context information. Currently, they implemented and deployed two applications: ActiveCampus Explorer, which uses students' locations to help engage them in campus life; and ActiveClass, a client-server application for enhancing participation in the classroom setting via PDAs. Like in MoCA, location is inferred by measuring the RF signals from 802.11 Access Points.

STEAM [11] 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.

YCab [2] is also a framework for development of collaborative services for ad-hoc networks. 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. Among the offered collaboration services, there is a chat, a shared white-board, and sharing of images (video-conferencing) and user files.

A common feature observed in most aforementioned environments 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. Of all the systems presented, only STEAM and ActiveCampus, use information about the current context (e.g. the location) for triggering appropriate adaptations of the application's behavior 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.

In MoCA we take a similar approach as in ActiveCampus, where context information (of any mobile user) may not only trigger user-transparent adaptations, but may also affect the specific functions available (and behavior) of the application at each point of time and space. Through its core services and the *ProxyFramework*, MoCA makes 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, and available resources, which she can use according to the specific needs of the application.

Compared to ActiveCampus' architecture, MoCA proposes a decentralized context-service service (CIS), which can be used by other services for deriving some higher-level and application-specific context information. Moreover, MoCA also supports service integration, extensibility and evolution through the Discovery Service and well-defined interfaces between the core services.

VI. 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* for WinXP, WinCE and Linux[1], the *Configuration Service*, and prototypes of the *Context Information Service* (CIS) and the *Location Inference Service* (LIS). Concerning the *ProxyFramework* , until now we have only a bare-bones implementation for W-Chat, which includes simplified versions of the components *Discovery*, *Caching Management* and *Context Management*, since only these were necessary 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. Application NITA is still being developed, but also here the advantage of using MoCA's LIS service will cause a significant reduction of complexity.

In 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.

Acknowledgments

This work is being supported by following research grants: FINEP/CNPq (number 552.068/02-0) and CNPq (number 55.2028/02-9).

References

[1] Paramvir Bahl and Venkata N. Padmanabhan. RADAR: An in-building RF-based user location and tracking system. In *INFOCOM (2)*, pages 775–784, 2000.

[2] D. Buszko, W.-H. Lee, and A.S. 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*, October 2001.

[3] M. Caporuscio, A. Carzaniga, and A. L. Wolf. Design and evaluation of a support service for mobile, wireless publish/subscribe applications. Technical report, Department of Computer Science, University of Colorado, January 2003. citeseer.nj.nec.com/565390.html (Last visited: November 2003).

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

[5] A. Carzaniga, D.S. 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] Guanling Chen and David Kotz. A survey of context-aware mobile computing research. Technical Report TR2000-381, Dept. of Computer Science, Dartmouth College, November 2000.

[7] P. Coschurba, K. Rothermel, and F. Dürr. A fine-grained addressing concept for geocast. In *Proc. International Conference on Architecture of Computing Systems, ARCS 2002, Karlsruhe*, volume 2299 of *LNCS*, April 2002.

[8] W. G. Griswold, R. Boyer, S. W. Brown, and T. M. Truong. A component architecture for an extensible, highly integrated context-aware computing infrastructure. In *Proc. of the 25th International Conference on Software Engineering (ICSE 2003), Portland, Oregon*, May 2003.

[9] 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.

[10] Cecilia Mascolo, Licia Capra, and Wolfgang Emmerich. *Advanced Lectures in Networking*, volume LNCS 2497, chapter Middleware for Mobile Computing (A Survey), pages 20–52. Springer Verlag, 2002.

[11] Rene Meier and Vinny 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.

[12] Daniela Nicklas, Matthias Grossmann, and Thomas Schwarz. Nexusscout: An advanced location-based application on a distributed, open mediation platform. In *Proc. 29th Int. Conference on VLDB, Belin*, September 2003.

[13] Howard Rheingold. *Smart Mobs: The Next Social Revolution.* Perseus Publishing, October 2002. ISBN: 0738206083.

[1]These implementations are mostly independent of the 802.11b chip set.