IEEE distributed systems ONLINE
Expert-authored articles and resources

IEEE DISTRIBUTED SYSTEMS ONLINE 1541-4922 © 2004 Published by the IEEE Computer Society
Vol. 5, No. 10; October 2004

MoCA: A Middleware for Developing Collaborative Applications for Mobile Users

Vagner Sacramento, Markus Endler, Hana K. Rubinsztejn, Luciana S. Lima, Kleder Gonçalves, Fernando N. Nascimento, Giulliano A. Bueno, *Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro*

MoCA—the Mobile Collaboration Architecture—is a middleware for developing and deploying context-aware collaborative applications for mobile users. It comprises client and server APIs, core services for monitoring and inferring the mobile devices' context, and an object-oriented framework for instantiating customized application proxies.

Portable computing devices with wireless communication interfaces—such as PDAs or smart phones—are becoming more powerful and commonplace. Consequently, demand has increased for applications and services that support communication and collaboration among mobile users. This new distributed computing environment poses new challenges, such as host mobility, limited device resources, and intermittent connectivity. However, it also opens up a range of different and unexplored forms of collaboration among mobile users, in which, for example, information about user locality and proximity could play a distinguished role in determining an interaction's form and participants.

We argue that collaboration in a static network differs significantly from collaboration in a mobile network. While for collaboration based on static networks, one implicitly assumes that all user devices have stable connectivity, this isn't the case in a mobile environment. Because mobile networks suffer from weak and intermittent connectivity, a user might become temporarily unavailable even though he or she is still engaged in the collaboration session. Hence, in a mobile setting, the requirements for synchronous views and mutual perception for the collaborating peers (collaboration awareness) needs to be redefined.

Another difference is related to user mobility. When users are mobile, the collaborating group

tends to be more dynamic and arises spontaneously, motivated by a shared common interest or situation. Moreover, the way a mobile user interacts with other community or group members tends to be more variable, asynchronous, and dependent on the user's current context, activity, or interest.

Finally, collaboration between mobile users usually isn't driven by a global and predefined goal or task, such as cooperative work on a digital or physical artifact. Instead, it's driven by spontaneous and occasional initiatives for mutual sharing of information, contribution to the development or improvement of public knowledge, and so on. This makes participation in a collaboration more spontaneous and irregular, and usually motivated by implicitly (or explicitly) assessed gain of reputation owing to the contribution of higher-quality, more precise, or more relevant information.[1]

All the aforementioned characteristics suggest that environments for developing mobile collaboration applications and services should incorporate new mechanisms that would facilitate collecting, aggregating, and accessing (at the application level) different kinds of information about the individual and collective contexts of collaborating users. This information can become directly available to the collaborating peers (for example, for enhancing the group's collaboration awareness) or can be used for adapting the application's behavior (for example, its available functions or user interfaces) to the current situation.

MoCA (*mo*bile *c*ollaboration *a*rchitecture) is a middleware architecture for developing context-processing services and context-sensitive applications for mobile collaboration. Our work on this architecture is part of a wider project that aims to experiment with new forms of mobile collaboration and implement a flexible and extensible service-based environment for developing collaborative applications for infrastructured mobile networks.

MoCA overview

We designed MoCA for infrastructured wireless networks. The current prototype works with 802.11 wireless networks, but it will also be possible to adapt the architecture for cellular data network protocols, such as General Packet Radio Service.

The MoCA infrastructure consists of client and server APIs, basic services supporting collaborative applications, and a framework for implementing application proxies (the ProxyFramework). Each application has three types of elements: servers, proxies, and clients. The first two execute on nodes of the static network, while clients run on mobile devices. The proxy intermediates all communication between the application servers and the clients. Applications with requirements to scale to numerous clients might have several proxies

executing on different networks. An application's proxy could be in charge of several tasks, such as adapting the transferred data—for example, applying data compression, protocol conversion, encryption, user authentication, context processing, service discovery, handover management, and others. Most of these tasks require significant processing effort, so the proxy is also a way to distribute the application-specific processing among the server and its proxies.

A collaborative application's server and client should be implemented using the MoCA APIs because they hide from the application developer most details concerning the interactions with the services that the architecture provides (which we discuss in the next section). We designed the APIs and the basic services to be generic and flexible, so they'd be useful for different types of collaborative applications—for example, those based on synchronous or asynchronous, message- or sharing-oriented communication.

The ProxyFramework—a white-box framework[2]—can be used to create application proxies according to the collaborative application's specific needs. It facilitates access to MoCA's basic services and the programming of application-specific adaptations triggered by events related to context changes.

In MoCA, most of these tasks require significant processing effort, so the proxy is also a way to distribute the application-specific processing among the server and its proxies.

Additionally, the architecture offers the following components and core services, which support the development of context-aware collaborative applications.

The *monitor* is a daemon executing on each mobile device. It collects data concerning the device and network's state and sends this data to the *context information service* (CIS) executing on one or more nodes of the wired network. The collected data includes the wireless quality, remaining energy, CPU usage, free memory, current access point (AP), and the set of APs with their corresponding signal strengths that are within the mobile device's range.

The CIS is distributed, and each of its servers receives and processes the mobile device's context information sent by the corresponding monitors. It also receives requests for notifications (that is, subscriptions) from application proxies and generates and delivers events to them whenever a change in a device's state is of interest to this proxy.

The *discovery service* (DS) stores information—such as name, properties, addresses, and so on—about any application (that is, its servers and proxies) or any service registered with the MoCA middleware.

The *configuration service* (CS) stores and manages configuration information for all mobile

devices so that they can use MoCA's core services, such as the CIS and the DS. The CS stores the configuration information in a persistent *hash table*, where each entry (indexed by the mobile device's media access control address) holds the CIS and DS servers' addresses (IP and port), and the periodicity by which the monitor must send the device's information to the CIS. The media access control address-specific indexing is essential for implementing a distributed CIS, where each server gets approximately the same context processing load.

The *location inference service* (LIS) infers a device's approximate geographical location by comparing the device's current radio frequency (RF) signal pattern (received from all "audible" 802.11 APs) with the signal patterns previously measured at predefined reference points in an indoor or outdoor area, using a similar algorithm as in the RADAR project.[3] For this, the LIS uses the device's context information collected by the CIS, and measures the mean value (10 probes) of the Eucledian distance to each reference point. Because the RF signal is subject to much variation and interference, the location inference is only approximate. Its precision depends on the number of audible APs and reference points in the area. The LIS lets the administrator define symbolic regions of arbitrary size and rectangular shape and a hierarchical description of regions and nested subregions.

Registering and executing a collaborative application

Figure 1 shows the typical sequence of interactions among MoCA's elements, illustrating the roles these elements play during registration and execution of a collaborative application.

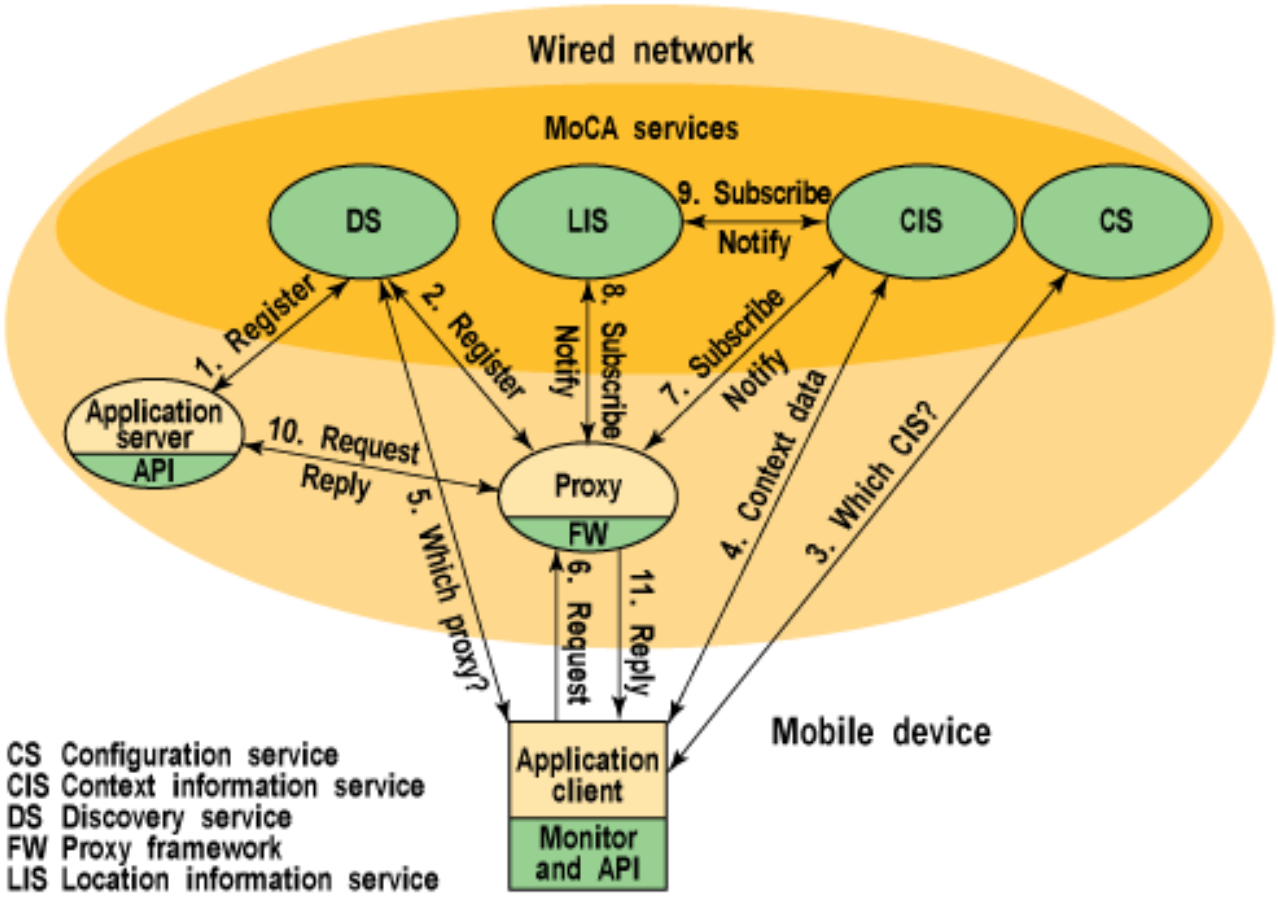

Figure 1. A typical interaction sequence between a collaborative application and MoCA's core services.

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 to discover how to access a given collaborative service in their current network —usually through the "closest" 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 we mentioned, the monitor obtains the address of the target CIS and the periodicity for sending the context information from the CS when it is started (Step 3). Thereafter, the monitor periodically sends the device's context data to the CIS (Step 4).

After discovering a proxy that implements the desired collaborative service through the DS (Step 5), the client can start sending requests to the application server. Each request gets routed through the corresponding proxy (Step 6), which processes the client's request, applying the application's specific adaptations, and forwards it to the application server. For example, the proxy might subscribe at the CIS with an *interest expression* (Step 7), registering its interest in notifications about context changes of a particular client (device). An interest

expression could be, for example, {"FreeMem < 10KB" OR "APChange = True"}, which would generate a notification if either the device has little free memory or has switched the access point.

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 so, the CIS generates a notification and sends it to all proxies that have registered interest in this change of the device's state.

Applications that require location information register their interest with the LIS (Step 8), which in turn subscribes to the CIS (Step 9) to receive periodic updates of the device's RF signals. The LIS uses these signals 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 back to some or all proxies (Step 10), which can then adapt or process the reply (for example, compress it, filter it, and so on) according to the new state of the corresponding mobile device on its wireless connection. Such context-specific adaptation depends on the collaborative application's specific requirements. For example, if the proxy is informed that the quality of a mobile device's wireless connectivity 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, such as the figures, apply some compression to the data, and so on. Moreover, the proxy could use other context information, such as the device's location, to determine when and how data should be delivered to the client (Step 11).

The architecture also implements mobility transparency for the applications. When a mobile device 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 the proxy is available, the collaboration session state is transferred to this new proxy.

Programming support

As we mentioned, MoCA provides server and client APIs and the ProxyFramework to implement an application.

Server API

The server API offers some interfaces that handle the server's configuration and

communication options. Through primitive `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 using `protType` and can be done through either an event service (using primitives `subscribe/publish`) or sockets (using primitives `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.

Primitive `init(serviceName, properties, protType)`, configures the application client so that it can use the services MoCA offers. It also starts the monitor (as a daemon). Moreover, it queries the DS to discover all the available proxies for the corresponding application. It does this by sending the application or service name, its properties, and the specific protocol to be used when interacting with the proxy (for example, short message service, User Datagram Protocol, Transmission Control Protocol, Java Messaging Service, or Wireless Application Protocol). With this information, the client can select the corresponding implementation of the client API's primitives (`send/recv`, `publish/subscribe`). Through the `getStatus()` interface, the client can query the context information collected by the monitor, which the application can use to trigger a local adaptation on the client side. This interface returns a reference to an XML-based description of the context information.

ProxyFramework

MoCA offers an object-oriented framework that supports the development of the application's proxy. This framework gives the application developer simple mechanisms for accessing context information related to client devices that interact with the server through the proxy and for defining how the proxy should adapt to changes in the client's context.

The ProxyFramework offers such facilities through common features and design patterns that can be found in most distributed applications that use a proxy to cope with device mobility and intermittent connectivity. The framework consists of interfaces for a set of concrete components (frozen spots), which implement the proxy's common and predefined functionalities, and interfaces for a set of abstract components (hot spots), which can be implemented and extended according to each application's specific demands.

Context-aware mobile collaboration

We have developed two context-aware collaborative applications as MoCA case studies. W-Chat (wireless chat) simply uses the connectivity information of a device's context, while NITA (Notes in the Air) uses location information inferred by the LIS.

W-Chat

W-Chat is a simple chat program distinguished by its ability to diffuse the connectivity status of chat room participants and support "conversational catch up" after a temporary device disconnection—that is, the user gets all messages posted during the disconnection time. To implement these functions, W-Chat uses connectivity information from the CIS.

W-Chat's proxy intercepts all messages (commands and events) from the client to the server and vice versa. It also has a local message buffer for each client, so when a user reconnects to the network, W-Chat's client and proxy synchronize their states, and the user gets the set of most recent messages from each chat room in which he or she was participating. The proxy also registers at the CIS its interest in any disconnection and reconnection event from any of its client's devices.

When any W-Chat client disconnects—for example, when having moved to a region without wireless coverage—a characteristic icon showing this new connectivity status appears close to the user's name in the list of chat room participants. This additional information about mutual availability for collaboration (a new form of collaboration awareness) helps users decide if they should expect immediate messages from other users. We implemented W-Chat in Java 2 Micro Edition (J2ME).

NITA

NITA is an application for posting text messages (and files in general) to a symbolic region as if it were a chat room. So, any user that's currently in (or that enters) this region and is properly authorized will automatically receive these messages. In the literature we can find several other projects with similar services combining messaging with spatial events.[4,5] However, most of these services were implemented from scratch or without a general middleware support for context monitoring and inference.

In NITA, when sending a message, a user can specify its destination (a symbolic region), names of the users authorized to read it, and how long the message will be readable. Moreover, a client can search for available NITA servers, their regions, and visible users in each of those regions. Potential receivers can set their visibility flag (on/off), choose which types of messages they want to receive, and choose whether to immediately display the

message or log it for future reading.

Because NITA is essentially a message retrieval service driven by spatial events (device *X* detected in region *Y*) and communication events, it interacts closely with the LIS.

The NITA proxy is responsible for querying the LIS about the area's structure and registering interest in its clients' location changes. Moreover, it manages the client's profile—that is, it decides whether to filter out some messages, log the messages, or forward them to the client. Although the NITA server could perform many of these tasks as well, moving these tasks to the proxy was necessary for scalability purposes.

Conclusion

As we mentioned, this work is part of a wider project that aims to investigate collaboration support for mobile users. Collaboration among mobile users requires new and different middleware services than the ones traditional groupware provides for wired networks. In particular, we believe that not only a user's individual context information (such as his or her location or connectivity) but also collective context information (such as users' proximity to each other) can help enrich collaboration awareness and also allow for new forms of collaboration that haven't yet been explored in conventional, wired collaboration.

Compared with other middleware and environments for mobile collaboration (see the "Related Work"), MoCA offers a generic and extensible infrastructure for developing both new services for context acquisition and inference, and collaborative applications that use this context information to determine the form, contents, and participants of a collaboration.

So far, we've implemented the monitor for WinXP and Linux (mostly independent of the 802.11b chip set), the configuration service, and prototypes of the CIS and LIS. Concerning the ProxyFramework, so far we have only a bare-bones implementation for W-Chat that includes simplified versions of the discovery, caching-management, and context management components because these are necessary for W-Chat.

During the development of the W-Chat application, we could perceive the benefits of using MoCA's services, APIs, and the ProxyFramework, which considerably reduced the application's complexity. Also, for the NITA application, we noticed that using MoCA's LIS significantly facilitated its development.

In another research thread, we're investigating ways to define user interests using ontologies and designing a service for evaluating user affinity and discovering similar (or

complementary) interests. Our goal is to design collaborative applications that use information about both user proximity and interest affinity to determine the participants of a collaboration.

Acknowledgments

The following research grants support this work: Brazilian National Research Funding Agency (CNPq) project numbers 552.068/02-0 and 5.2028/02-9.

References

1. H. Rheingold , *Smart Mobs: The Next Social Revolution,* Perseus, 2002.
2. M. Fayad and D.C. Schmidt , "Object-Oriented Application Frameworks,"*Comm. ACM,* vol. 40, no. 10, 1997, pp. 32–38.
3. P. Bahl and V.N. Padmanabhan , "RADAR: An In-Building RF-Based User Location and Tracking System,"*Proc. 19th Ann. Joint Conf. IEEE Computer and Comm. Socs.* (INFOCOM 2000), vol. 2, IEEE CS Press, 2000,pp. 775–784.
4. D. Nicklas , M. Grossmann and T. Schwarz , "NexusScout: An Advanced Location-Based Application on a Distributed, Open Mediation Platform,"*Proc. 29th Int'l Conf. Very Large Data Bases* (VLDB 2003), Morgan Kaufmann, 2003, pp. 1089–1092.
5. P. Coschurba , K. Rothermel and F. Durr , "A Fine-Grained Addressing Concept for Geocast,"*Trends in Network and Pervasive Computing: Proc. Int'l Conf. Architecture of Computing Systems* (ARCS 2002), LNCS 2299, Springer-Verlag, 2002, pp. 101–113.

Vagner Sacramento is a PhD candidate in the Department of Informatics at the Pontifícia Universidade Católica in Rio de Janeiro (PUC-Rio). His research interests include distributed systems and protocols, mobile computing, middleware, and network security. He received his MSc in computer science from UFRN in 2002. Contact him at vagner@inf.puc-rio.br.

Markus Endler is an assistant professor at PUC-Rio's Department of Informatics. His main research interests are distributed algorithms and systems, mobile and ubiquitous computing, and middleware for mobile networks. He received his Dr.rer.nat. in computer science from the Technical University in Berlin and the title "Professor Livre-docente" from the University of São Paulo. He is member of the ACM and the Brazilian Computer Society (SBC). Contact him at endler@inf.puc-rio.br.

Hana K. Rubinsztejn is a PhD candidate in PUC-Rio's Department of Informatics. Her current research interests are mobile and ubiquitous computing and middleware for distributed and mobile systems. She received her MSc in computer science from the University of Campinas. Contact her at hana@inf.puc-rio.br.

Luciana S. Lima is a PhD candidate in PUC-Rio's Department of Informatics and a researcher at the PUC-Rio's TeleMídia Laboratory. Her main research interests are quality of service, mobile and ubiquitous computing, and context-aware computing. She received her MSc in computer science from PUC-Rio. Contact her at slima@inf.puc-rio.br.

Kleder Gonçalves is an MSc candidate at PUC-Rio's Department of Informatics. His current research interests are mobile and ubiquitous computing and software engineering. He received his BSc in computer science from Universidade Federal do Pará. Contact him at kleder@inf.puc-rio.br.

Fernando N. Nascimento is an MSc candidate at PUC-Rio's Department of Informatics. His main research interests are middleware for distributed systems, component-based systems and mobile computing. He received his BSc in computer science from the Federal University of Rio Grande do Norte. Contact him at ney@inf.puc-rio.br.

Giulliano A. Bueno is a third-year undergraduate student at PUC-Rio's Department of Informatics. His current research interests are mobile and distributed computing. Contact him at giubueno@lac.inf.puc-rio.br.

Related Work

Researchers have done considerable work related to middleware and programming environments for mobile and context-aware applications,[1,2] and much of this research has influenced our work. However, due to space limitations, we only discuss architectures and environments with similar goals to ours.

YACO (Yet Another Collaboration Environment)[3] is a framework for collaborative work based on SIENA ,[4] a distributed, content-based publish-subscribe communication infrastructure, and on MobiKit,[5] a mobility service toolkit based on proxies. Using MobiKit's operation `moveOut`, a client can inform a server of its disconnection. Whenever the client

reconnects to the network, it can invoke operation `moveIn`, which lets it replay all the events missed during the time period it was disconnected. As a collaboration environment, YACO offers a message service, a service for user discovery, and a service for sharing 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 publish-subscribe. MOTION provides teamwork services for managing user groups and access rights (through its DUMAS subsystem), artifact storage and sharing, and support for different mobile devices.

ActiveCampus[7] is a large project at the University of California, San Diego, that provides an infrastructure that focuses on integrating 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) that supports a clear separation of the context information's collection, interpretation, association with physical entities, and service-specific representation. The project researchers have 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 classrooms via PDAs. As with MoCA, ActiveClass determines location by measuring the radio frequency signals from 802.11 access points.

STEAM [8] is event-based middleware for collaborative applications where location plays a central role. The system is specially designed for ad hoc mobile networks, so it's inherently distributed. It supports filtering of event notifications on the basis of both subject and proximity.

YCab[9] is also a framework for developing 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 communication, the client component, and its state. Among the offered collaboration services is a chat, a shared white-board, image sharing (video-conferencing), and file sharing.

Most of the aforementioned environments try to shield all aspects regarding mobility and user location from the application developer, aiming for a seamless, anywhere-available service. Of all the systems presented, only STEAM and ActiveCampus use information about the current context (for example, location) to trigger appropriate adaptations of the application's behavior or enable context-specific application functions, such as selecting collaboration partners on the basis of proximity or disseminating mobile devices' connectivity status.

MoCA's approach is similar to that of ActiveCampus, where (any mobile user's) context

information might not only trigger user-transparent adaptations, but could also affect the specific functions available (and the behavior) of the application at each point in time and space. Through its core services and the ProxyFramework, MoCA makes available to the application developer a wide range of context information—for example, the user device's (approximate) location, the connectivity quality, the device characteristics, and available resources—which he or she can use according to the application's specific needs.

Compared to ActiveCampus's architecture, MoCA provides a decentralized context information service, which other services can use to derive 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.

References

1. G. Chen and D. Kotz, , *A Survey of Context-Aware Mobile Computing Research,* tech. report TR2000-381, Dept. of Computer Science, Dartmouth College, 2000.
2. C. Mascolo , L. Capra, and W. Emmerich, , "Mobile Computing Middleware,"*Advanced Lectures on Networking: NETWORKING 2002 Tutorials,* LNCS 2497, Springer-Verlag, 2002, pp. 20–52.
3. M. Caporuscio and P. Inverard, , "Yet Another Framework for Supporting Mobile and Collaborative Work,"*Proc. 12th IEEE Int'l Workshops Enabling Technologies: Infrastructure for Collaborative Enterprises* (WETICE 03), IEEE CS Press, 2003, pp. 81–86; http://citeseer.nj.nec.com/583641.html.
4. A. Carzaniga , D.S. Rosenblum, and A.L. Wolf, , "Achieving Scalability and Expressiveness in an Internet-Scale Event Notification Service,"*Proc. 19th Ann. ACM Symp. Principles of Distributed Computing,* ACM Press, 2000, pp. 219–227.
5. M. Caporuscio , A. Carzaniga, and A.L. Wolf, , *Design and Evaluation of a Support Service for Mobile, Wireless Publish/Subscribe Applications,* tech. report, Dept. of Computer Science, Univ. of Colorado, 2003; http://citeseer.nj.nec.com/565390.html.
6. E. Kirda , et al., "A Service Architecture for Mobile Teamwork,"*Proc. 14th Int'l Conf. Software Eng. and Knowledge Eng.,* ACM Press, 2002, pp. 513–518.
7. W.G. Griswold , et al., "A Component Architecture for an Extensible, Highly Integrated Context-Aware Computing Infrastructure,"*Proc. 25th Int'l Conf. Software Eng.* (ICSE 2003), ACM Press, 2003,pp. 363–372.
8. R. Meier and V. Cahil, , "Exploiting Proximity in Event-Based Middleware for Collaborative Mobile Applications,"*Proc. Distributed Applications and Interoperable Systems: 4th IFIP WG6.1 Int'l Conf.* (DAIS 03), LNCS 2893, Springer-Verlag, 2003, pp. 285–296.
9. D. Buszko , W.H. Lee, and A.S. Helal, , "Decentralized Ad Hoc Groupware API and Framework for Mobile Collaboration,"*Proc. Int'l ACM SIGGROUP Conf. Supporting Group Work,* ACM Press, 2001,pp. 5–14.

Cite this article: Vagner Sacramento, Markus Endler, Hana K. Rubinsztejn, Luciana S. Lima, Kleder Gonçalves, Fernando N. Nascimento, and Giulliano A. Bueno, "MoCA: A Middleware for Developing Collaborative Applications for Mobile Users," *IEEE Distributed Systems Online*, vol. 5, no. 10, 2004.