Support for Context-Aware Collaboration

Hana K. Rubinsztejn, Markus Endler, Vagner Sacramento,
Kleder Gonçalves, Fernando Nascimento
e-mail: {hana,endler,vagner,kleder,ney}@inf.puc-rio.br

PUC-RioInf.MCC34/04 September, 2004

Abstract

This paper describes a middleware architecture with its location inference service (LIS), and an application for context-aware mobile collaboration which is based on this architecture. 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, in particular their location. This architecture is suited for the development of new kinds of collaborative applications in which the context information (connectivity, location) plays a central role in defining both the group of collaborators, and the communication mode.

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

Resumo

Este artigo descrevemos uma arquitetura middleware, seu serviço de inferência de localização (LIS) e uma aplicação para colaboração móvel sensível a contexto baseada nessa arquitetura. A arquitetura, denomindada *Mobile Collaboration Architecture* - MoCA provê APIs cliente e servidor, um conjunto de serviços básicos para monitorar e inferir o contexto de dispositivos móveis, em particular sua localização. Essa arquitetura é adequada ao desenvolvimento de novos tipos de aplicações colaborativas nas quais a informação de contexto (conectividade, localização) representa o papél central na definição do grupo de colaboração e do modo de comunicação.

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

Support for Context-Aware Collaboration

I. Introduction

With the increasing popularity of mobile technologies, high-speed wireless communication is now available in many locations such as corporate offices, factories, shopping malls, university campi, airport halls, cafes and at homes. Moreover, with the widespread availability of cheaper and more powerful portable devices, there is also an increasing demand for services that support seamless communication and collaboration among mobile users. User mobility has also inspired the development of many *location-based* services, such as automated tourist guides or generic location-specific information services. We believe that user mobility and wireless communication capabilities also open a wide range of new and yet unexplored forms of collaboration, in which information about the user's context, for example, her position, plays a central role in defining both the group of collaborators, and the communication mode.

Concerning the first aspect, we believe that, unlike in traditional groupware, the group of collaborators tends to be dynamic and formed spontaneously, for example, motivated by a common interest, situation or environment shared among the peers. In particular, collaboration happens in form of spontaneous and occasional initiatives of requesting or giving assistance, sharing news or contributing to the building of public knowledge. Thus, participation is very often motivated by the implicit gain of reputation caused by providing help or a contribution [1].

In terms of the second aspect, we think that the user's context (e.g. her current location, quality of the connectivity, or her current activity/task) will be an important factor when determining the most appropriate communication mode for a given situation or task, i.e. whether synchronous or asynchronous communication is to be used in each situation. In particular, for mobile collaboration we are convinced that both modes of communication are equally important and that they will probably be used in an interleaved way.

Finally, due to the intrinsic weak and intermittent connectivity in mobile networks, there is need to re-define the notion of mutual *collaboration awareness*. For example, when a user is engaged in a synchronous collaboration session and for any reason suffers an involuntary and temporary disconnection, in most cases it is important that her new context (e.g. disconnected) is shared among the collaborating peers, so as to minimize the disruption of the group dynamics. Hence, we think that mobile collaboration tools should support some degree of context sharing.

In order to share our vision of new forms of context-aware collaboration, consider the following two scenarios:

> **Scenario 1:** While in his office, Bob is co-editing a document with colleagues using a groupware tool at his desktop connected to a 100 Mbps network. At a certain point he has to leave for a departmental meeting (in the same building), but wants to continue the collaboration, during the meeting. He then decides transfer the collaboration session to his WiFi-enabled PDA, where he has a PDA-ported version of the same groupware application. On his way to the meeting he keeps the device on, but obviously cannot engage in any serious interaction until he arrives in the meeting room. There, his visualization and interaction capabilities are seriously affected, due to device limitations (smaller screen and lack of keyboard), the smaller network bandwidth of WiFi, as well as his limited attentiveness. Therefore, Bob decides to focus on the exchange of messages with his co-workers, giving them instructions on what he would like them to do. However, because he is still on-line, none of them accepts his requests, arguing that he should do it by himself.

This scenario suggests that the users of a collaboration application would have concrete benefits if their application were able to notify all the session participants whenever Bob changed his context: e.g. when Bob

transferred the session to this PDA, when he was walking, or entered the meeting room. If they had this information, they would probably know what to expect from Bob, and would better understand his change of behavior.

> **Scenario 2:** In the cafeteria during lunch, Alice realizes that she has probably forgotten her home keys in the lab, where she had been in the morning. But since she has an important meeting afterward she cannot return there to check for the keys. So she calls the lab but because it is lunch hour, nobody picks up the phone. And since she rarely visits the lab in the afternoon, she does not know who could be there later. So she decides, using her wireless enabled PDA, to send a message to the lab's virtual note board, asking for any student or lab staff who happens to come into the lab to store away her keys in her drawer. As soon as Peter enters the lab and turns on his notebook, he is notified of a new message from Alice to the lab. Because he is a regular lab user, he is authorized to open it. He reads the message, finds and stores Alice's keys, and closes the message with an OK. When doing so, Alice is automatically notified that someone has read her message and probably attended her appeal. Although she does not know who it was, she knows that if necessary she could later check in the virtual note board who it was. So she decides to remotely remove her note. She then rushes to the meeting, where she arrives 20 minutes late. In the meeting, she realizes that the most important topic has already been discussed. Instead of asking (and disturbing) someone, through her PDA, she searches for any meeting-specific note being shared by any of the participants. Fortunately, Helen the secretary, is her friend, and Alice can immediately know what she has missed. Moreover, she can see that right now Helen is scribbles: "Late again, uh?".

This scenario shows the benefits of location-specific asynchronous and synchronous communication. The first is useful as a form of any-casting, while the latter may be useful in situations where sharing of documents is required among people which are co-located, and participating in the same event, such as a meeting, a presentation or a class.

The aforementioned discussion and scenarios 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 group of users, 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 (MoCA) with its location inference service (LIS), and an application for context-aware mobile collaboration which is based on this 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 following session we describe the main design principles MoCA's support for application programming and the network requirements of our architecture. Section III presents a general overview of MoCA, its main components and their interactions. Our current work on context-aware collaborative applications is presented in Section V. In section VI we discuss some related work and make a comparison with the MoCA architecture. Finally, in section VII, we make some considerations with regard to the MoCA properties, and mention other ongoing work that in the scope of this project.

II. Design Principles

The *Mobile Collaboration Architecture (MoCA)* 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 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 execute several tasks, such as adaptation of the transferred data, e.g. compression, protocol conversion, encryption, user authentication, processing of context information, service registration and discovery, 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.

When designing MoCA, we adopted following principles:

Scalability: The architecture accounts for the possibility of implementing any service as a pool of servers, guaranteeing distribution and location transparency to the clients. New servers can be added to the system according to the service demand.

Extensibility: Nearly all of MoCAs services can be extended, incorporating new functionality, and new services can be included independently of the existing ones. For example, the location service discussed in Section IV can be regarded as an additional optional service for location-aware applications.

Simplicity: MoCA facilitates the use of its services by the application developer. Using the *ProxyFramework* and the APIs for the Server and the Client, the developer has easy access to all basic services (and other services) without having to care about the details of their execution and interactions, and can therefore focus on the application's logic.

Heterogeneity: Several communication protocols can be used for the services of the architecture. For example, the proxy of an application can use several protocols, such as (JMS, TCP, UDP, SMS, WAP, etc.) to interact with different clients executing on mobile devices prepared for different communication technologies, such as CDMA, GSM/GPRS, 802.11.

Flexibility: With the goal to offer a flexible and robust execution environment, all the services in MoCA shall be modular and distributable, easy to configure, update and maintain. Moreover, application servers and clients should be executable both on mobile and stationary devices. For example, a server for sharing files (e.g. music files in a peer-to-peer application) could either execute on a notebook with wireless interface or on a workstation.

Independence: Yet another design principle was to depend only on the most basic and widely available internet protocols and services. Related to this, there are just three basic requirements of using MoCA for developing and executing a collaboration application:

- The application developer must use the *ProxyFramework* and the client and server APIs provided by the architecture in order to be able to use the MoCA services;
- TCP/IP must be available in the wired network
- DNS (*Domain Name System*) must be available in the wired network, since it is used by the services to resolve network names.

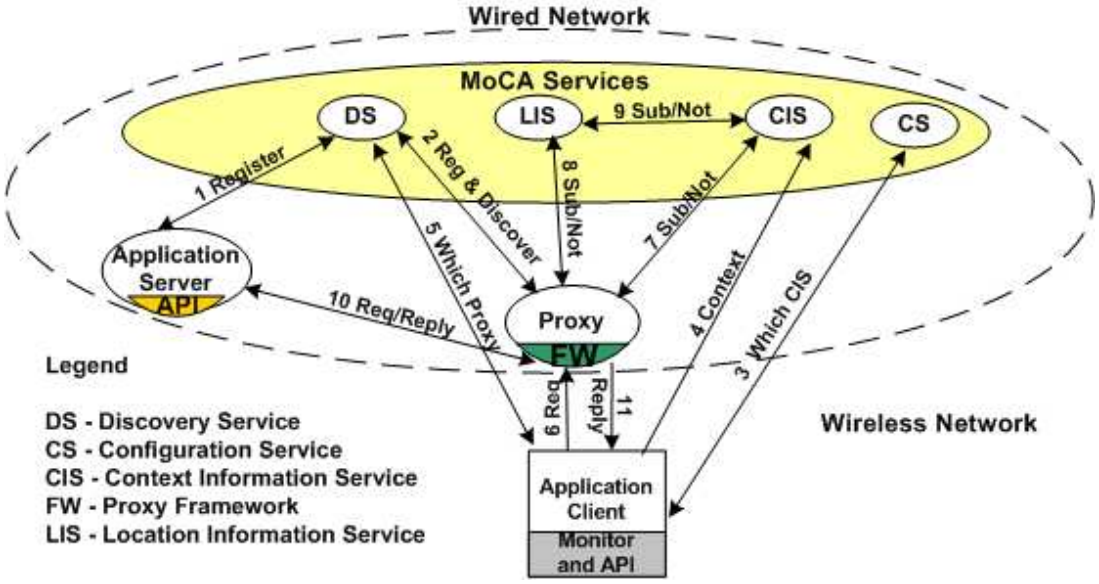

Fig. 1. Typical Interaction Sequence between a collaborative application and MoCA's core services

III. Overview of MoCA

The 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.

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 by 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 *symbolic* location of a device. Details of this service will be presented in Section IV.

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 *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.

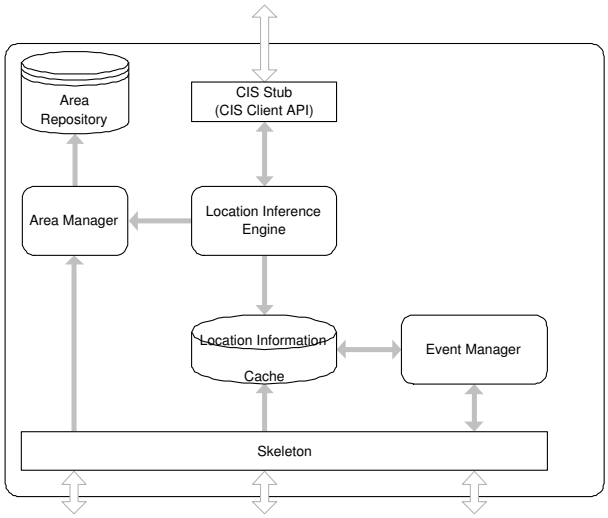

Fig. 2. LIS Architecture

IV. Location Inference Service

The *Location Inference Service (LIS)* is responsible for inferring the approximate location of a mobile device from the *raw* context information collected by the CIS for this 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. Therefore, before being able to make any inference, the LIS database has to be populated with RF signal probes (device pointing in several directions) at each reference point, and inference parameters must be chosen according to the specific characteristics of the region. In fact, the number of reference points determines the reliability of the inference. LIS uses a hybrid and hierarchical location model where position can be given either by coordinates or by symbolic name.

The LIS architecture is outlined in Figure 2.

The `CIS stub` requests to the CIS periodic notifications of the RF signal strengths sensed by a device from several APs (i.e. the RF pattern), collects and prepares this information for location inference.

The `Inference Engine` applies the *Multiple Nearest Neighbor* technique, as used in RADAR [2], to infer the approximate location of the device. This technique calculates the Euclidean *Signal Distance* d_k to a reference point k, based on the difference between each component $i \in [1, m]$ of the RF signal pattern (i.e. signal from each one of the m APs) collected for the device s'_i, and the corresponding component of the RF pattern s_{ki} of the reference point, which has been previously measured and recorded.

$$d_k = \sqrt{(s_{k1} - s'_1)^2 + (s_{k2} - s'_2)^2 + ... + (s_{km} - s'_m)^2}$$

Each coordinate of the (estimated) position of the device is then computed as the weighted mean value of the corresponding coordinate of only some reference points, which happen to have the lowest Signal Distances.

Since the RF signal is subject to much variation and interference, the inference can only be approximate. However, in order reduce the error, when comparing the RF patterns of a device and a reference point, we always use the mean value of several (e.g. $N = 20$) probes, and extreme values are previously discarded.

As part of LIS' configuration, a MoCA administrator can define geographic regions of arbitrary size and shape, assign them a *symbolic region* name, and use them to construct a hierarchical topology (i.e. composite

regions with nested sub-regions) for his site. This information will be stored in the `Area Repository` in XML format. Accessing the `Area Manager`, the `Inference Engine` is able to map the coordinates of the inferred point to a symbolic region.

The result of an inference is the information of a device's coordinates, or if it is positioned inside any of the symbolic regions defined by the user. This information will then be recorded in the `Location Information Cache`. Because of the hierarchical topology, even if a device is not detected within an atomic symbolic region (e.g. a room), it will be detected in the enclosing region.

The `Event Manager` is responsible for processing queries and requests for notifications from applications which are interested in the location (or its change) of one or several devices, as well as sending the notifications whenever a device's location is updated in the `Location Information Cache`. LIS provides an interface for both device-based and region-based queries and notification subscriptions. Within LIS, each device is identified through its MAC Address.

Preliminary tests have shown that LIS can deliver satisfactory precision. We tested it on the 600 m^2 region (5th floor of our CS building), where signals from up to eight 802.11 APs can be sensed. Within this region, we defined reference points (i.e. measured and recorded their RF signal patterns) at approximately every 4 meters, mainly in corridors and common halls. The measurements were made at several, randomly chosen, test points in this region. For each test point, we made 20 measurements and computed the error (the Euclidean Distance, in meters) between its actual geographic position and the position inferred by LIS.

TABLE I

PRELIMINARY PRECISION RESULTS

Percentage of Test Points	50%	70%	90%
Error (in meters)	1.56	1.74	2.84

Table I presents the errors (in meters) obtained with 50%, 70% and 90% of the test points. For example, for 70% of the tests, the error was at most 1.74 meters. And even for 90% of the tests, the error at most 3 meters.

The results obtained in our experiments are similar to the ones described in [2], which was expected since the algorithms are equivalent. As a next step we will experiment with other inference algorithms, such as the ones described in [3], [4] that use a probabilistic approach which seems to be more robust to variations of the RF signal strength. However, for the sort of location-aware applications we are developing, the achieved accuracy is fairly good and sufficient.

V. APPLICATION: NOTES IN THE AIR

In this section we present a Collaborative Application which we are developing as case study of MoCA.

NITA (from "Notes in the Air") is an application to post text messages (and files, in general) to a symbolic region. Hence, any client which is currently in (or enters) this region and has the proper authorization will automatically receive this message. There are several other projects with similar services combining messaging with spatial events. 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 symbolic 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.

When a message is first read, its author receives an acknowledge message with a timestamp on it. It does not contain the reader's identity because such information would raise privacy problems, i.e., the author would know what time the reader was in a location. The acknowledge message is just used to inform the sender that her message was read at least once, and when.

When a user enters in a new symbolic region, the LIS service informs the NITA proxy about this change of user location. The latter forwards this information to the server application which calls the *removeUser()* method on the *Location object* related to the previous region and the *addUser()* method on the *Location object* representing the user's current region. Then, the *Location object* sends to the client application on the mobile device the list of IDs of all messages posted to the location it represents. A *new messages on the air* warning appears on user's screen, and the user can open a window to browse the list, showing subject and author of each message. If she decides to read one, the message's ID is sent to the application server, which retrieves the related message from the database and sends it to the client. Hence, only the desired messages are actually sent to the users. Besides, the user can choose whether to save or discard a message. On the former case, the message is saved on a previously configured URL, which can point to a local file system or to the user's remote account on the NITA server.

Besides this asynchronous kind of communication, NITA also provides a synchronous mode of communication. Each symbolic region has a chat room associated to it, allowing users to send/receive synchronous messages to/from people in the same location. This feature is interesting for regions with many people. For instance, a conference room, where users can communicate with each other without disturbing the speaker, or a bar, where shy people want to first have a virtual conversation before engaging in a meeting in the physical world.

Furthermore, the user does not need to be in the same place as her peers. When she first opens the application, she is automatically added to her current location and the chat room related to it. But she can browse and join other chat rooms and therefore be able to communicate with peers in other regions. In this case, her icon appears in a different way so the others know that she is not physically present in the region. It is also possible to send messages to a specific peer, as in normal peer-to-peer communication.

In addition to symbolic regions, a chat room can be created and associated to subjects. Hence, users can meet to talk about a topic of their interest, no matter where they are. This way, NITA works like a conventional wireless chat application.

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 III), 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.

VI. Related Work

Much research related to middleware and programming environments for mobile and context-aware applications [5], [6] has been done, and many influenced our work. However,we will only discuss some architectures/environments with similar goals as MoCA. Furthermore, we will present also some applications which use the device location as the main context information, like NITA.

A. Middlewares

ActiveCampus [7] 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.

Aura[8] is a project at Carnegie Mellon University which is developing a system architecture, algorithms, interfaces and evaluation techniques needed for pervasive computing environments. The architecture components comprise Coda - a nomadic file system, Odyssey - for resource monitoring and adaptation, Spectra - remote execution mechanism and Prism - a new task layer above applications that is responsible for capturing and managing user intent to provide support for proactivity and self-tuning. Aura also offers a bandwidth advisor service (for IEEE 802.11 wireless networks) with two components for monitoring and prediction, and a user location service based on signal strength and 802.11 access point information, similar to LIS.

STEAM [9] 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 [10] 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. Only STEAM, ActiveCampus and Aura 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-information 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.

B. Applications

There are several location-aware applications described in literature. comMotion[11], [12] and Stick-e Notes[13] post messages (to-do lists, news from the Internet, etc) to a place, using a GPS device to get the

user's location information. As in NITA, information is delivered according to physical location and/or user identity. Cyberguide[14] is a mobile context-aware tour guide that offers information to users according to their position or orientation. The information is retrieved by anyone who is in a specific area, and cannot be filtered according to user identity.

In many applications, the positioning information is only used by the beholder of the device and is not shared. This way, it doesn't contribute to other purposes, i.e., knowing the location of a peer might suggest places of potential interest, as mentioned in Conference Assistant[15], where users attending a conference indicate their level of interest in a particular presentation to the application. Some privacy issues arise from this feature, and therefore some applications, like Cricket[16], chose not to provide it, avoiding a common database that holds such information, as used in NITA. But we believe that with some precaution, like using buddy-lists and configuration of a set of properties, privacy can be assured.

According to the limits imposed by their positioning technology, all applications can be used either indoors (Cricket) or outdoors (comMotion, Stick-e Notes) but not both. Currently, NITA is also restricted to indoor use (because of LIS), but we designed NITA to be independent of the location technology. Furthermore, all the above-mentioned applications support either synchronous or asynchronous communication, but not both, as NITA does.

VII. 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 of both new services for collecting and/or processing context information, and collaborative applications that make use of this information to determine the form, the contents and/or the peers involved in the collaboration. By developing the LIS service, we have shown how MoCA can be extended to incorporate new services that infers an abstract context, as for example, the device location from "raw"context data, such as 802.11 RF signals.

So far, we have implemented the *Monitor* for WinXP and Linux[1], the *Configuration Service*, and prototypes of the *Context Information Service* and the *Location Inference Service*. Although, preliminary tests have shown that LIS can deliver satisfactory precision, we are still experimenting with different inference algorithms, as well as assessing the accuracy of the inference in other areas on the campus.

Application NITA is still being developed, but we can already notice the benefits of using MoCA's services, especially LIS, the APIs and the *ProxyFramework*. Their use reduced considerably the complexity of the application development since the application just needs to register for notification of context changes. We have already developed one other context-aware application using MoCA, named W-Chat which is a chat tool providing collaborative peers with information about mutual (wireless) connectivity. Moreover, we are also planning to use MoCA and LIS for the development of location-aware information services for the university campus.

[1]These implementations are mostly independent of the 802.11b chip set. And we are currently working on a WinCE version.

In another thread of research, we are investigating means of defining user interests using ontologies, and designing services for matchmaking of interests and skills. The goal is to design collaborative applications which use both information about user location and interest affinity for selecting the peers and the form of collaboration.

REFERENCES

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

[2] P. Bahl and V. N. Padmanabhan, "RADAR: An in-building RF-based user location and tracking system," in *INFOCOM (2)*, 2000, pp. 775–784. [Online]. Available: citeseer.nj.nec.com/bahl00radar.html

[3] A. M. Ladd, K. E. Bekris, A. Rudys, L. E. Kavraki, D. S. Wallach, and G. Marceau, "Robotics-based location sensing using wireless ethernet," in *Proceedings of the 8th annual international conference on Mobile computing and networking*. ACM Press, 2002, pp. 227–238.

[4] T. Roos, P. Myllymaki, H. Tirri, P. Misikangas, and J. Sievanen, "A probabilistic approach to wlan user location estimation," *International Journal of Wireless Information Networks*, vol. 9, no. 3, pp. 155–164, July 2002.

[5] G. Chen and D. Kotz, "A survey of context-aware mobile computing research," Dept. of Computer Science, Dartmouth College, Tech. Rep. TR2000-381, November 2000. [Online]. Available: ftp://ftp.cs.dartmouth.edu/TR/TR2000-381.ps.Z

[6] C. Mascolo, L. Capra, and W. Emmerich, *Advanced Lectures in Networking*. Springer Verlag, 2002, vol. LNCS 2497, ch. Middleware for Mobile Computing (A Survey), pp. 20–52.

[7] S. W. B. W. G. Griswold, R. Boyer 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.

[8] D. Garlan, D. Siewiorek, A. Smailagic, and P. Steenkiste, "Project Aura: Toward Distraction-Free Pervasive Computing," *IEEE Pervasive Computing*, no. 2, pp. 22–31, April-June 2002.

[9] 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.

[10] 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.

[11] N. Marmasse, "comMotion: a context-aware communication system," in *CHI '99 extended abstracts on Human factors in computing systems*, MIT Media Laboratory. ACM Press, 1999, pp. 320–321.

[12] N. Marmasse and C. Schmandt, "Location-Aware Information Delivery with ComMotion," in *HUC - Handheld and Ubiquitous Computing*, ser. Lecture Notes in Computer Science, vol. 1927. Springer, September 2000, pp. 157–171.

[13] J. Pascoe and N. Ryan, "Stick-e notes," http://www.cs.ukc.ac.uk/research/infosys/mobicomp/Fieldwork/Sticke.

[14] S. Long, R. Kooper, G. Abowd, and C. Atkeson, "Rapid prototyping of mobile context-aware applications: The cyberguide case study," in *2nd ACM International Conference on Mobile Computing and Networking (MobiCom'96)*, November 1996.

[15] A. K. Dey, D. Salber, G. D. Abowd, and M. Futakawa, "The conference assistant: Combining context-awareness with wearable computing," in *ISWC - 3rd International Symposium on Wearable Computers* , 1999, pp. 21–28.

[16] N. Priyantha, A. Chakraborty, and H. Balakrishnan, "The Cricket Location-Support System," in *6th ACM/IEEE MobiCom*, Boston, MA, August 2000, pp. 32–43.