

A Middleware Architecture for Context-Aware and Location-Based Mobile Applications

José Viterbo¹, Vagner Sacramento², Ricardo Rocha¹, Gustavo Baptista¹,
Marcelo Malcher¹ and Markus Endler¹

¹Department of Informatics
Pontifícia Universidade Católica do Rio de Janeiro
R. Marquês de São Vicente, 225
22453-900, Rio de Janeiro, Brasil
{viterbo, rcarocha, gbaptista, marcelom, endler}@inf.puc-rio.br

²Institute of Informatics
Universidade Federal de Goiás
Bloco IMF I
74001-970, Goiânia, Brasil
vagner@inf.ufg.br

Abstract

The development of location and context-aware applications is greatly facilitated by the use of context-provisioning middleware. However, development of such applications still remains a challenge from the point of view of software engineering. In this paper we present MoCA, a service-oriented middleware architecture that supports the development and deployment of distributed context-aware applications for mobile users. Besides explaining its main services and APIs, we discuss in which ways the MoCA architecture supports some well-known software engineering principles that apply to the design and implementation of context-aware applications. Furthermore, we give an overview of its usage and present the most notable prototype applications that have been developed on the top of MoCA.

1 Introduction

One of the key requirements of distributed applications that run on mobile devices is their ability to perceive — and opportunistically adapt to — the conditions of their physical and networked execution environments, in order to optimize their operations and deliver the best possible service to the users. However, implementing the so called context-

awareness¹ is inherently complex due to the specificity and the heterogeneity of the executing environments (e.g., various execution platforms, wireless networks and sensor technologies). Hence, development of such applications may be greatly simplified by the use of context-provisioning middlewares [28]. The main reason is that the acquisition, distribution, storage and management of context data becomes transparent to the application programmer, who can focus on implementing the application's logic. Nevertheless, despite the huge amount of publications describing elaborated context models [29] and extensible frameworks or middleware systems [1], unfortunately, to date there are only very few freely available, and — in fact — easily usable systems for the development of context-aware mobile applications.

In this paper, we report our experience in developing and effectively using the *Mobile Collaboration Architecture* (MoCA) [26], a stable service-oriented middleware architecture that has been used by several research groups in Brazil and abroad for the development of small-scale or experimental context-aware applications.

MoCA is a service-based architecture which offers support for the development of distributed context-aware applications for mobile devices interconnected through IEEE 802.11 wireless LANs. MoCA's services and components provide means of collecting, distributing and processing

¹In the remainder, we will consider location-awareness as a specific kind of context-awareness.

context data obtained directly from the mobile devices, i.e., the state of the devices' resources, as well as parameters of the wireless network connection. In addition, MoCA makes available to the application developer a set of APIs for synchronous and asynchronous access to context data and other context-specific services.

In the next section we discuss the goals and requirements that guided the MoCA's design and implementation. In Section 3 we give a bird-eye's view of MoCA's architecture, its basic services and the set of context data that it delivers, some optional services and MoCA's personalities. Then, in Section 4 we present the main APIs and the typical use for context-awareness in MoCA-based applications. Section 5 briefly discusses how MoCA meets some well-known principles of system's software engineering, facilitating the development of context-aware mobile applications. Section 6 then presents a few selected context-aware application prototypes that have been developed over the years using MoCA. Section 7 compares MoCA with other context provisioning middleware platforms commonly referenced in literature. Finally, Section 8 draws some conclusions and points to future developments.

2 The Design and Implementation of MoCA

It has been well recognized that the development of context-aware and -adaptive mobile applications is a complex task and requires the careful observance of several well-known software engineering principles [9, 22]. MoCA was designed as a layered architecture following a *context server* approach [1], in which largely independent services provide an infrastructure for collecting, distributing and processing context information. This approach aimed at facilitating the development of applications by observing principles such as separation of concerns, multi-level abstractions, incremental development, flexibility of customization and multi-language and interoperability support.

Our main goal with the MoCA project was to develop an extensible architecture and a coherent set of efficient services for the collection and distribution of system's context and location information of mobile devices, such as handhelds, tablet PCs or notebooks. The main motivation was to free the application programmer from the burden of programming the 'low-level' interfaces (with the device platforms) for probing the system's raw context data, and the basic services for storing context data and implementing context distribution mechanisms.

While MoCA's independent services permit separation of concerns, the simple and comprehensive set of APIs available offer multi-level abstractions for the application developer to easily use these services and fully concentrate on the application's logic and adaptation requirements. Furthermore, the service-oriented architecture is extensible al-

lowing not only flexibility of customization — i.e., the selection of a specific set of available services to be deployed and used —, but also the incremental development of additional context producing and processing services that may be necessary for a particular application. All these features are further discussed in Section 5.

Since the early publications [24, 26], several service components have been incrementally added and improved, so as to extend the middleware's facilities [25, 30]. Over the last four years, several students and researchers have used MoCA for building their experimental context-aware mobile or ubiquitous applications [5, 6, 18], and to our satisfaction, most of them appeared to be fairly satisfied. This has been confirmed by the statements and evaluation grades given by some developers to several aspects of the architecture such as ease of installation and use, online documentation, robustness and reliability.

3 System Overview

The typical architecture of a MoCA based application using the client/server paradigm is shown in Figure 1, where the MoCA basic services are represented, i.e., the services that implement the main functionalities of the architecture, such as context management and location inference. The figure shows also some optional services, i.e., services that are used for some specific applications. In its most general form, an application based on MoCA is composed by one or more application servers, which execute on static machines in the wired network, and the application clients, which execute on mobile devices, such as Smart Phones, TabletPCs or Notebooks. The application server is usually the client of the MoCA services, i.e. a consumer and point of access to context information of all the mobile devices². Details about all the MoCA services are presented in the following subsections.

3.1 Basic Services

MoCA's *Monitor* is the component responsible for probing several device and network interface resources, converting them into a standard format and normalized scale, and making this context information available both to the local application client and to other instances of the application. It must be executed on each mobile device. The set of produced context information includes the quality of the wireless connectivity (in terms of RF signal strength – RSSI), the current Access Point Id (AP Id), the current IP address and mask, the device's remaining energy level, CPU usage and free memory space, as well as the list of all APs within

²Of course, the mobile clients can access context information directly, in a way similar to the application servers.

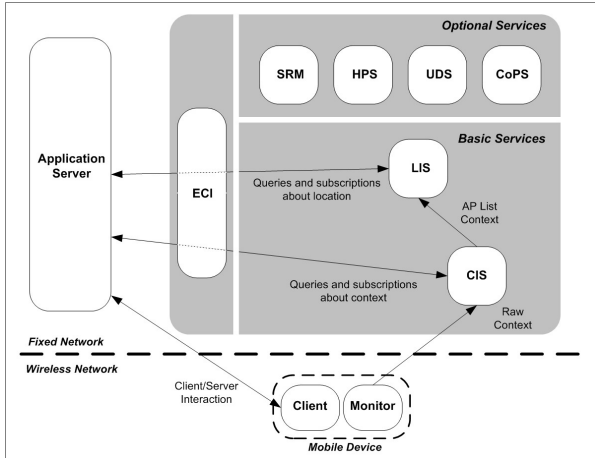


Figure 1. Typical architecture of a MoCA-based client/server application.

the range of wireless access of the device with their corresponding sensed RSSI or the GPS position data, if available. The two last ones are used for location inference.

This set of context information is periodically sent by the *Monitor* to the *Context Information Service* (CIS). This service — which may be implemented as a single or multiple CIS servers — is responsible for collecting, storing and processing this context data. All context variables — or tags (shown in Table 1) — are indexed by the MAC address of the mobile device. Hence, by a simple mapping function the range of possible MAC addresses can be equally split among the CIS servers, achieving some load balance and scalability of this service. Using the same mapping function, the applications (e.g. client or server components) can request CIS for context information from any of the monitored devices, both synchronously or asynchronously.

Through synchronous requests (via request-reply messages) it is possible to obtain the latest context information about a device. And through asynchronous requests (via subscription and notifications) applications can register at CIS their interest in a specific context state, which is described by a logic expression involving several context variables (or tags) of the target device. For example, with the expression $\{\text{FreeMemory} < 10\text{MB}\} \text{ OR } \{\text{APChange} = \text{True}\}$ as a context subscription, an application component expresses interest to be notified by CIS whenever the amount of free memory of a given device drops below 10 MB *or* the device has performed a handover between access points.

The context data made available by CIS can be also used by other MoCA services to derive higher-level context information. This is the case of MoCA's *Location Inference Service* (LIS), which uses the context information about the

RF signal strengths of nearby IEEE 802.11 APs (i.e. CIS' *APList*) to infer the approximate location of each mobile device [20]. This is done by evaluating the similarity between the RSSI distribution pattern currently sensed by the device and the RSSI distribution patterns measured at specific *reference points* in the geographic region of interest (*indoors* or *outdoors*). LIS delivers location information in terms of *symbolic regions* (e.g. a room, a hall, a corridor, a building floor, or a street section) that are relevant to the applications, instead of geographic positions.

As with CIS, an application can interact with LIS to access the location information both synchronously and asynchronously. Using the first mode, it can be informed of the symbolic region in which a specific device is currently located, or which are all the devices located in a specific location. This mode is useful for easily identifying co-located devices, i.e. users. Using the asynchronous mode, the application may get notifications each time a target device enters/leaves some region, or whenever the set of devices of a specific symbolic region changes. Moreover, the application can query LIS to learn all the symbolic regions that have a mapping (e.g. have been previously measured) in the service.

Another basic building block of MoCA is the *Event Communication Interface* (ECI), an interface for Publish/Subscribe communication that supports a simple *subject-based* subscription mode, but provides several configurable modes of context-oriented notifications: *One-time*, *N-time* or *Periodic* notification, as well as notifications whenever an application's context-interest expression switches from valid to non-valid, and vice-versa [2]. The ECI interface is used for implementing the interactions with the MoCA services, but can, as well, be used by the application developer that needs to implement asynchronous communication within its mobile application.

3.2 Optional Services

MoCA allows not only the creation of *symbolic regions* of arbitrary size and shape but also their aggregation into a hierarchical structure defining nested sub-regions. This functionality is implemented through an auxiliary service named *Symbolic Region Manager* (SRM), which provides an interface to create, manage and request information about hierarchies of *symbolic regions*. In these hierarchies *composite symbolic regions* are described as an aggregation of *atomic symbolic regions* (determined by LIS) or other *composite symbolic regions*. For instance, *composite symbolic region* "Computer-Building/1stFloor" may be defined as containing *atomic symbolic regions* "room10", "room11", "room12" and "hall".

Although MoCA was originally designed to support only symbolic positioning based on comparison of IEEE 802.11

Context Tags	Type	Description
CPU	Integer	CPU usage (0 to 100%)
EnergyLevel	Integer	Remaining battery level (0 to 100%)
AdvertisementPeriodicity	Integer	Frequency of Monitor context sending (sec.)
APMacAddress	String	MAC Address of the current AP
FreeMemory	Long	Total amount of free memory (KB)
DeltaT	Long	Time interval since previous delivery of context data (msec.)
OnLine	Boolean	True iff device is connected to any AP
IPChange	Boolean	True iff device switches IP address
APChange	Boolean	True iff device switches the AP
APlist	String	List of tuples (AP Id, RSSI)
GPSPos	String	Pair (Lat, Long)

Table 1. Context information tags provided by CIS.

RF signal strength patterns (i.e. RSSI fingerprint), we have extended it to process also GPS-based positioning information for mobile devices with GPS capabilities. For this purpose, we developed the *Hybrid Positioning Service* (HPS), which is responsible for integrating symbolic and geographical positioning of a mobile device. HPS gets either the symbolic location of a device from LIS, whenever this information is available, or the geographical coordinates from CIS, when the device can provide GPS coordinates. When provided only with symbolic location it uses a database with vectorial geographical data to convert symbolic location to geographical coordinates (considering the centroid of the *symbolic region*). Conversely, if only GPS data is available, it uses the same database to map the geographic coordinate into a symbolic location, e.g., a polygon representing a specific area. Therefore, HPS is capable of delivering both the symbolic and the geographic position of a client.

In mobile systems, some applications may be interested in using the resources or services that are available in the user’s vicinity or in the currently visited network domain, such as a nearby printer. On the other hand, some services may be available only to clients within a well defined region, e.g. within a building or campus. Considering these aspects, we implemented the *Ubiquitous Discovery Service* (UDS) for the MoCA architecture [30]. This service enables application clients to search for a service whose *scope of availability* — i.e. the symbolic region where a service is available — matches the client’s location. Furthermore, UDS offers a notification service that makes possible for client applications to register their interest in services with given characteristics and scope, and to be asynchronously notified when a desired service becomes available at the client’s physical location.

Finally, another optional service of MoCA is the *Context Privacy Service* (CoPS), which allows the users to define and manage her privacy policies concerning the disclosure of their device’s context information [25].

3.3 Personalities

MoCA services and APIs were originally implemented specifically to support the development of applications in Java. In order to extend the usability of MoCA’s context services to other programming languages we developed what we call three *MoCA Personalities*: MoCA/WS, MoCA/ORB and MoCA/MAX [31]. These personalities materialize as proxies that implement the mapping of MoCA’s Java-based APIs (i.e. CIS and LIS clients) to other implementation highly implementation technologies such as Web Services or Multi-agent Systems. By using such proxies the application developer can then implement her mobile application using different programming language and have access to context and location information both in the synchronous and asynchronous modes. It turned out that this multi-language support of our system was essential for the development of some application prototypes.

4 Using MoCA

The MoCA services and APIs (developed in Java) and the MoCA *Monitor* (developed in C ANSI) may be freely downloaded from the MoCA Project site³. Until now, we have implemented versions of the *Monitor* for Windows XP, Windows Mobile, Linux and Symbian (for the Smart Phone Nokia S60). The *Monitor* was implemented in C because Java does not provide APIs to access some low-level drivers. The MoCA services and APIs may be grouped in three different sets. The first set consists solely of the *Monitor*, which must be installed only at the mobile device. The second set comprises the packages that provide the basic services, such as ECI, CIS and LIS. Those services must be installed on the machines that are part of the infrastructure environment, generally in the wired network. The third set is formed by the APIs used to implement the context-aware

³ <http://www.lac.inf.puc-rio.br/moca>

applications, which must be installed on the machines that are part of the development and execution environment.

4.1 Application Programming Interfaces

The MoCA APIs may be divided in three groups: (1) the communication APIs, which provide interfaces for synchronous and asynchronous (publish/subscribe) communication using either TCP or UDP; (2) the main APIs that provide interfaces for accessing the MoCA basic services; and (3) the optional APIs that provide interfaces for accessing each of the optional services. The features of the communication and main APIs are presented in the following:

- *Communication Protocol* - used for developing applications that implement synchronous communications using either TCP or UDP messages. This API is used for most MoCA services;
- *Event-based Communication Interface* - implements asynchronous communications (publish/subscribe), allowing clients, called subscribers, to register interest for specific events and to be notified when such events happen. This is a general purpose API and is also used by all applications that communicate with some MoCA services, like CIS, LIS and SRM;
- *CIS Client* - provides a communication interface with CIS, allowing applications to execute synchronous and asynchronous queries to get context information about devices;
- *LIS Client* - provides a communication interface with LIS, allowing applications to execute synchronous and asynchronous queries to get location information about devices and symbolic regions;

Using the *CIS Client* API, an application may get updated context information about a given mobile device. Figure 2 shows in Lines 1 to 11 a synchronous query to a CIS server expecting at IP address “139.82.24.239” to get context information about mobile device with MAC address “00:02:2D:A5:06:46”. Using asynchronous communication, this same API allows applications to subscribe at the CIS server registering interest in a specific context state of the device — described by SQL expressions correlating context variables —, to be notified whenever this state holds. In the asynchronous interaction shown in Figure 2, Lines 12 to 15, the application requests a specific CIS server running at IP address “139.82.24.239” to be notified whenever the remaining energy level of the mobile device with MAC address “00:02:2D:A5:06:46” comes below 30% of its capacity or the free memory is less than 10 Mbytes. When an application needs to subscribe at (or query) a pool

```
01 InetSocketAddress server = new InetSocketAddress("139.82.24.239","55001");
02 InetSocketAddress local = new InetSocketAddress("localhost", "5000");
03 Request request = new Request("00:02:2D:A5:06:46");
04 tcpClient = new TCPConnection();
05 tcpClient.open(server);
06 tcpClient.send(request);
07 reply = (Reply) tcpClient.nonBlockingReceive(3000);
08 tcpClient.close();
09 ctx = (ContextInformation) reply;
10 deviceCTX = ctx.getDvcContext();
11 DeviceCtxManagement.printOutDeviceContext(deviceCTX);
12 CisSubscriber subscriber = new CisSubscriber(ECIClient.TCP_server, local);
13 Topic topic = subscriber.subscribe("00:02:2D:A5:06:46","(EnergyLevel < 30) OR
(FreeMemory < 10)");
14 CISListener listener = new CISListener("low resources");
15 subscriber.addListener(listener, topic);
```

Figure 2. Code for interaction with CIS server

of CIS servers, instead of a single one, a high-level *subscribe* function is used, where the IP of the server is determined by the adequate MAC-mapping function.

```
01 LocationInferenceService lis = null;
02 lis = new LocationInferenceService("localhost", "55021", "55020", "5000", "TCP");
03 allregions = lis.getAtomicRegions();
04 String [ ] areas = new String [allregions.length];
05 for (int i = 0; i < allregions.length; i++) areas [i]= allregions [i].getName();
06 alldevices = lis.getDevices();
07 region = lis.getRegion("00:02:2D:A5:06:46");
08 devices = lis.getDevices("Room 201");
09 DeviceListen deviceListen = new DeviceListen();
10 lis.subscribe("00:02:2D:A5:06:47", deviceListen);
11 RegionListen regionListen = new RegionListen();
12 lis.subscribe("Room 202", regionListen);
```

Figure 3. Code for interaction with LIS server

Using the *LIS Client* API, an application may execute synchronous queries to LIS to get information about a specific mobile device or a specific *symbolic region*, or — using asynchronous communication — subscribe to LIS for being notified whenever a given device changes location or whenever any device enters or leaves a given *symbolic region*. Figure 3 shows synchronous queries to a LIS server running at IP address “localhost” to get the list of all symbolic regions mapped by the service (Line 3), the list of all devices being monitored by the service (Line 6), the symbolic region in which the mobile device with MAC address “00:02:2D:A5:06:46” is located (Line 7), and the list of devices which are located in the symbolic region named “Room 201” (Line 8). The Code shows also the application issuing subscriptions to register interest in location-change events for the device with MAC address “00:02:2D:A5:06:46” (Lines 9 and 10), and device-change

events (devices entering or leaving) for the symbolic region named “Room 202” (Lines 11 and 12).

Each of the optional services (SRM, HPS, UDS and CoPS) has its own APIs which are very similar to those discussed for CIS and LIS, i.e., they allow synchronous and asynchronous communication and facilitating the access to each service’s functionalities.

5 Leveraging some Software Engineering Principles

As the development of context-aware applications requires the observance of several well-known software engineering principles [9], some of these principles guided our design and development of the MoCA architecture itself and, therefore, were fundamental to the success of the project. We believe that our current experience with the implementation of several application prototypes already provides some insights of the practical implications of the use of our middleware architecture to mobile applications development. In this sense, this section presents an initial discussion on how MoCA meets some of these well-known principles of system’s software engineering, and how by such it facilitates the development of context-aware mobile applications.

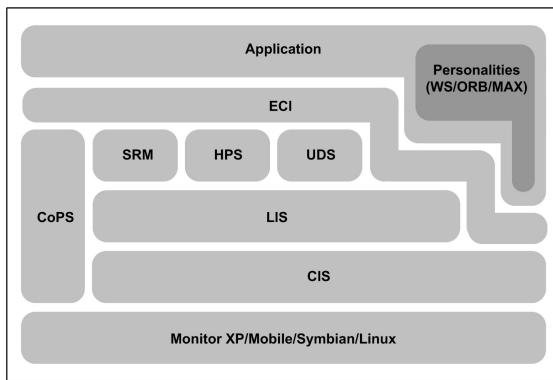


Figure 4. Layered architecture of MoCA

Figure 4 shows in detail the MoCA layered architecture, which was designed observing, among others, the following software engineering principles that we discuss below: separation of concerns, multi-level abstractions, incremental development, flexibility of customization and multi-language and interoperability support.

Separation of concerns is absolutely necessary to cope with the complexity of designing a distributed mobile applications. MoCA addresses this question by making context access transparent to the application developer. It provides a coherent and intuitive interface to access system’s context and location information, as presented in Section 4. There-

fore, by using the MoCA CIS and LIS services and APIs the developer neither has to deal with the intricacies of implementing the modules for probing resources or network data (for the mobile devices and their network interfaces), nor to implement a location-sensing software based on RF strength and AP visibility. Although this does not account for the full separation of concerns, in fact it is a very important component of it, saving much of the developer’s time.

Middleware hides the low-level resources but makes explicit the key concepts involved in the development of mobile applications, e.g., the management of location data, event notification, quality of service assessment, adaptability, etc, in the form of *abstractions*, which are the natural means by which separation of concerns is realized, allowing application developers to have the appropriate level of knowledge about the computational scenario [22]. Since MoCA was designed as a general-purpose middleware system, it is clear that its abstractions should be general, and not application specific. However, in order to support developers of applications with different context-awareness demands, MoCA provides different levels of abstraction, ranging from the complex notion of a hierarchy of *symbolic regions* down to the simple context value change event. At the high-level end, LIS and SRM support abstractions related to location inference: *symbolic regions* and hierarchy of *symbolic regions*, respectively. While both are very useful for implementing location-aware applications, at the low-level end CIS’s `APMacAddress` attribute can be regarded as a simpler and less precise location information. Another useful abstraction is *network connectivity*, given by CIS’s `Online` attribute and/or the current APs RF signal strength, which may, for example, be used to initiate some kind of message buffering at the application server. Yet another abstraction is the information of a recent address change by the device (e.g. CIS’s `APChange` or `IPChange`, which may be used to trigger some resource reservations at application servers or proxies.

Incremental development is yet another fundamental engineering principle that is intrinsically supported by MoCA architecture. MoCA services are largely independent of each other and, therefore, allow the application developer to evolve her application gradually by using — and creating — more services as needed. For example, an application may first use only CIS’s information of the current AP to estimate the device’s location, then it may use LIS for obtaining a more precise location information, and finally, the developer may create an application-specific location hierarchy for SRM and use it for a larger region of interest. Furthermore, MoCA’s general-purpose Pub/Sub component ECI and its simple and extensible Monitor/CIS protocol (based on attribute-value strings) facilitates the inclusion of new context sources (such as GPS coordinates) and the development of application-specific context-processing services

which use some basic context data already provided to infer higher-level context information. For example: one could implement a context-aggregating component that evaluates the number of mobile devices with same `APMacAddress` (provided by CIS), and implements a service to balance the load among the WLAN APs (i.e. forcing the reconnection of some application clients to another AP with less served devices). We may also cite HPS as an example of incremental development in the implementation of MoCA architecture, since it extends the functionalities of LIS to provide symbolic location from GPS coordinates.

Flexibility of customization is motivated by the well-acknowledged requirement of configuring the middleware system only to the application's specific needs. MoCA supports a flexible combination of its services. This is made possible through the loose coupling of its services. For example, some applications may need to deploy only CIS and the Monitor at the devices, while others may require Monitors, CIS and LIS for location-awareness. In addition, the application developer may choose to deploy the discovery service UDS to enable clients to discover and connect to other application servers.

Multi-language and interoperability support is necessary, since mobile applications may bear components implemented in different programming languages or using different communication standards. MoCA supports this interoperability by its *Personalities*, MoCA/ORB and MoCA/WS, which are proxies that enable access CIS and LIS as a CORBA-based service or a Web Service. Both proxies export exactly the same operations as the original CIS and LIS Java interface, and thus enable both the synchronous and asynchronous access modes. Through the use of these MoCA personalities, context-awareness is made available also to application clients implemented in other languages, such as C++ or C#.

6 Applications

Several context-aware applications have been developed using the MoCA architecture. In this section we present the most representative applications that use MoCA to implement context-aware and/or location-based functionalities, such as Nita, iPH, UHS and UbiQuiz.

6.1 Notes in the Air

The Nita (*Notes in the Air*) [10] allows the publication of messages (and files in general) in *symbolic regions*, as if they were virtual boards. Thus, any user who is (or enters) in a symbolic region to where a message was sent — and has the proper authorization — will be notified about this message and will be able to read it and save it on her device. This application also allows the synchronous location-based

communication, i.e., the creation of chat rooms defined for *symbolic regions*. As such, only the users that are inside a given *symbolic regions* (for example, a classroom) will be able to participate in a chat. And everytime a user leaves that region (physically), he will be removed from the respective room.

Nita is implemented as a client/server application in which the Nita server subscribes LIS to get location information about the devices where Nita clients are running on. Hence, the Nita server is notified about any location change of the devices, being able of correctly delivering the messages and managing the chat rooms associated with *symbolic regions*.

6.2 Interactive Presenter for Handhelds

The *Interactive Presenter for Handhelds* (iPH) [19] is a distributed application that supports the sharing and co-edition of slide presentations. The application may be executed not only on resource-full devices such as notebooks and tablet PCs, but also on handhelds running Windows Mobile. During a collaborative session using iPH, one of the participants plays the role of the instructor, the one that controls the presentations, selecting and broadcasting the slides and asking to the other participants to give contributions at particular points of the presentation. Each participant may contribute by making changes on his copy of the slides and sending it to the instructor. Eventually, the instructor may choose some of the contributions to be displayed to all students (through the projector-host), in order to discuss a given solution and stimulate the participation of other students in the classroom.

In order to ease and improve the user experience, iPH was also implemented as a context-aware system — it may adapt its functionalities according to context information. iPH requests system context information (e.g., the device's energy level, free memory, quality of the wireless connection, etc.) and location to MoCA to help users to connect to a classroom-specific collaborative session, or to adapt some of its functionalities according to it, such as enabling or disabling some collaboration capabilities. As iPH was fully implemented in C#, it uses MOCA/WS — one of the MoCA personalities described in Subsection 3.3 — to communicate with the MoCA services (CIS and LIS). For example, the instructor may determine that only devices with a minimum amount of free memory and low CPU usage may join a collaborative session or that only students/devices within the classroom should be capable of giving contributions. Then if the free memory of a device is below an indicated threshold or the CPU usage is above some expected value, it will not be admitted for a session. On the other hand, whenever a device is detected outside the specific classroom the "submit" button at the contributor's GUI will be disabled,

but it will return to normal state whenever the device is again detected inside that classroom.

6.3 Ubiquitous Health Services

Ubiquitous Health Services (UHS) [6] is a health services network that allows associated physicians to access *electronic patient records* (EPRs) from anywhere in a network that connects several associated hospitals. Physicians that collaborate with some of the associated hospitals can access the available services from different hospitals or from their homes, cars or their own offices, for example, using different types of wireless networks (GPRS, 3G, WiFi) and devices. Each hospital offers specific services to their collaborators and the network offers generic services to all collaborators and associated hospitals. The physician may start a session with one device and transfer it to another one during execution — in a operation known as *application roaming*. UHS not only guarantees the consistency for the session data during migration, but also assures that the process takes place with small latency. If the roaming process is interrupted unexpectedly, it guarantees that the interrupted session data can be saved and later retrieved from the same device or from another one used by the physician.

In UHS, the location information for the whole environment is managed by two services: the Local Location Service (LLS), for each hospital in the environment, and the Global Location Service (GLS), which consists of a central provider for the environment. LIS is used for providing to LLS the symbolic location of users (carrying mobile devices) inside closed environments, e.g., a hospital. When a user is moving inside a hospital while connected to the network, his symbolic location is stored in a table managed by the LLS specific for that hospital. A module named location monitor executes in each LLS sending events to the GLS, which stores location information collected from all associated environments (e.g. all associated hospitals).

6.4 UbiQuiz

UbiQuiz [8] is a simple location-aware quiz game that runs on Nokia S60 smart phones. In this game, a (*human*) player must answer some questions that depend on both his location and the level he has achieved in the game. The goal of the player is to go through a number of rooms, answering some questions in each room, while upgrading his level in the game, until he reaches the last level and wins the game. At this moment he gets the permission to create and post a new question for the game. Starting from *beginner* level, the player successively passes to the levels *intermediate*, *advanced*, *expert* and *winner*. There are two kinds of questions, the ‘easy’ ones, that are aimed at *beginner* and *intermediate* players, and the ‘difficult’ ones, aimed

at *advanced* and *expert* players. After correctly answering all questions as an *expert*, the player becomes a *winner*, and may post his own question.

The game application was implemented as an UbiQuiz server that communicates with a client application on the Smart Phone to send the questions and to collect the answers. UbiQuiz server subscribes to LIS for being continually notified about the location of each device. It also keeps track of the level of each player and depending on both pieces of information it selects the questions to be sent to the players.

7 Related Work

In general, the majority of the middleware architectures that have already been proposed follow a same layered conceptual framework comprising protocols and services for: sensing, raw data retrieval, preprocessing and storage/management of the contextual information. In this section we focus on making some comparisons concerning the context provisioning middleware platforms commonly referenced in literature taking into account their architectural design, context model, availability to be used in real scenarios and technical documentation.

In fact, as discussed in [1], the architectural design of each middleware depends on special requirements and conditions such as the location of sensors (local or remote), the amount of possible users (i.e., the desired or intended scalability), the available resources of the used devices (desktops or mobile devices) or the facility for further extension of the system.

Considering these issues, the Service-Oriented Context-Aware Middleware (SOCAM) [12], Context-Awareness Sub-Structure (CASS) [7] and Context Managing Framework [17] are centralized architectures that in general receive context data from distributed context sensors and offer it in mostly processed form to the clients. Context Broker (CoBra) [4] follows an agent-based approach and Gaia [23], Context Toolkit [27], CORTEX System [3], and Hydrogen [16] a peer-to-peer architecture. However, Context Toolkit uses a centralized discovery service where distributed sensor units (called Widgets), interpreters and aggregators are registered in order to be found by client applications. MoCA architecture follows a centralized approach in order to provide an infra-structure for building and rapidly prototyping context-aware mobile services. The centralized architecture has been chosen targeting practical and feasible support for context-awareness in infra-structured networks, rather than ad-hoc networks. As mentioned, problems of dependability and lack of scalability of the centralized approach can be solved by deploying a cluster of networked servers, as it is possible with MoCA’s Context Information Service (CIS).

Related to the context model, the Service-Oriented Context-Aware Middleware (SOCAM), Context Managing Framework and Context Broker use Ontologies (OWL/RDF) to represent and process the context [33]. Ontologies provide a rich formalism for specifying contextual information. The Context-Awareness Sub-Structure (CASS) and CORTEX System use relational data model while Hydrogen use a Object-oriented model. Similarly to Context Toolkit, MoCA Architecture uses *attribute-value tuples* as context model. This model does not offer a declarative semantics about the sensed context and limits the reasoning and knowledge sharing capabilities. However, in current version of MoCA, the *attribute-value tuples* model provided appropriate support for the development of many context-aware applications and offers simple abstractions that facilitate the understanding of MoCA's context access by the application developers.

Differently from most aforementioned architectures, MoCA is an active project with regular updates and additions, as well as constant assistance to its users. Moreover, it offers a set of integrated and robust services such as the CIS and LIS, as well as simple-to-use APIs that facilitate the development of location- and context-aware applications in IEEE 802.11 networks. All these services and APIs are available for download in the website MoCA Project and have a rich documentation explaining how to deploy and use them. Furthermore, there are also some MoCA-based applications available to be downloaded and tried.

8 Conclusion

In this paper we presented MoCA — an extensible middleware architecture for context-provisioning — and discussed how it has been used for the development of a number of distributed mobile application prototypes. Although MoCA already represents a valuable aid for the development of such applications, we are aware that it represents only a first step towards a more comprehensible and mature software engineering discipline. We understand that, in spite of the supported engineering principles, there remain many open challenges for the specification, design and implementation of systems that cope with context-aware adaptation.

For example, other middlewares offer other kinds of interesting programming abstractions that yield to transparency of the infrastructure mechanism for context acquisition and dissemination. PACE's preferences [13] and profiles [32] are examples of such abstractions. However, such abstractions are very limited to separate the application's business logic from its context-specific logic, because they are mostly limited to provide vertical separation of concerns, i.e. concerns among software layers, instead of among software modules.

Since most frameworks use events to notify applications about contextual changes, it becomes difficult to structure the application's code in modular components. Moreover, context-aware computing introduces some particular aspects, such as context privacy [25] and adaptations that depend on user interactions [13]. Some of those challenges can be partly overcome with the use of special-purpose programming language constructs for describing adaptations and context-specific adaptations, but these languages introduce new programming abstractions which require special training by the developer.

Context modeling is another interesting topic that connects software engineering and context-aware computing. Most work in this direction proposes ontologies to describe and reason about higher-level context information, such as user activities and intentions. However, studies have shown that current ontology-based approaches are not suited for general purpose modeling [14]. Moreover, the cost of ontology-based reasoning hinders its usage in large-scale context-aware scenarios. In fact, there seems to be a trade-off between complex modeling and efficiency of context-based reasoning.

Finally, we see also some big challenges on enabling context-aware applications to become ubiquitous. In this case, it does not suffice just to provide a distributed context middleware, but applications must also reinterpret context information and consistently execute their context-based adaptations across different middleware systems. Thus, ubiquitous context-aware applications call for interoperability solutions and efficient context information dissemination approaches. Recent researches in middleware (e.g. [11], [15] and [21]) have focused on these challenges.

References

- [1] M. Baldauf, S. Dustdar, and F. Rosenberg. A survey on context-aware systems. *Intl. Journal of Ad Hoc and Ubiquitous Computing*, 2(4):263–277, 2007.
- [2] G. Baptista, M. Endler, V. Sacramento, and H. Rubinsztein. Uma API pub/sub para aplicações móveis sensíveis ao contexto (in Portuguese). In *Proc. of the 1st Workshop on Pervasive and Ubiquitous Computing (WPUC), part of 19th Intl. Symposium on Computer Architecture and High Performance Computing (SBAC-PAD 2007)*, 2007.
- [3] G. Biegel and V. Cahill. A framework for developing mobile, context-aware applications. In *Proc. of the 2nd IEEE Intl. Conf. on Pervasive Computing and Communications (PerCom'04)*, pages 361–365, 2004.
- [4] H. Chen. *An Intelligent Broker Architecture for Pervasive Context-Aware Systems*. PhD thesis, Department of Computer Science, University of Maryland, Baltimore County, December 2004.
- [5] K. Damasceno, N. Cacho, A. Garcia, A. Romanovsky, and C. Lucena. Context-aware exception handling in mobile agent systems: the MoCA case. In *Proc. of the 2006 Intl.*

Workshop on Software Engineering for Large-scale Multi-Agent Systems (SELMAS '06), pages 37–44, 2006.

- [6] J. B. Diniz, C. Ferraz, and H. Melo. An architecture of services for session management and contents adaptation in ubiquitous medical environments. In *Proc. of the 2008 ACM Symposium on Applied Computing (SAC '08)*, pages 1353–1357, 2008.
- [7] P. Fahy and S. Clarke. CASS – A middleware for mobile context-aware applications. In *Proc. of the Workshop on Context Awareness (MobiSys 2004)*, 2004.
- [8] C. Felicíssimo, J. Viterbo, L. Valente, M. Endler, C. Lucena, B. Feijó, and J.-P. Briot. Supporting agents in intelligent environments with protocol information. In *Proc. of the 4th IET Intl. Conf. on Intelligent Environments (IE 08)*, Seattle, WA, USA, 2008.
- [9] C. Ghezzi, M. Jazayeri, and D. Mandrioli. *Fundamentals of software engineering*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1991.
- [10] K. Gonçalves, H. K. Rubinsztein, M. Endler, B. S. Silva, and S. Barbosa. Um aplicativo para comunicação baseada em localização (in Portuguese). In *Anais do Workshop de Comunicação sem Fio e Computação Móvel*, Outubro 2004.
- [11] M. Grossmann, M. Bauer, N. Honle, U.-P. Kappeler, D. Nicklas, and T. Schwarz. Efficiently managing context information for large-scale scenarios. In *Third IEEE Intl. Conf. on Pervasive Computing and Communications (PerCom 2005)*, pages 331–340, March 2005.
- [12] T. Gu, H. Pung, and D. Zhang. A service-oriented middleware for building context-aware services. *Journal of Network and Computer Applications*, 28(1):1–18, 2005.
- [13] K. Henriksen, J. Indulska, T. McFadden, and S. Balasubramaniam. Middleware for distributed context-aware systems. *Lecture Notes in Computer Science*, 3760:846–863, 2005.
- [14] K. Henriksen, S. Livingstone, and J. Indulska. Towards a hybrid approach to context modelling, reasoning and inter-operation. In *1st Intl. Workshop on Advanced Context Modelling, Reasoning and Management*, pages 54–61, March 2004.
- [15] C. Hesselman, H. Benz, P. Pawar, F. Liu, M. Wegdam, M. Wibbles, T. Broens, and J. Brok. Bridging context management systems for different types of pervasive computing environments. In *1st Intl. Conf. on Mobile Wireless Middleware, Operating Systems and Applications (MOBILWARE)*, 2008.
- [16] T. Hofer, W. Schwinger, M. Pichler, G. Leonhartsberger, J. Altmann, and W. Retschitzegger. Context-awareness on mobile devices - the Hydrogen approach. In *Proc. of the 36th Annual Hawaii Intl. Conf. on System Sciences (HICSS'03) - Track 9*, pages 292–302, 2003.
- [17] P. Korpiää and J. Mäntyjärvi. An ontology for mobile device sensor-based context awareness. In *Modeling and Using Context*, pages 451–458. Lecture Notes in Computer Science 2680, February 2003.
- [18] F. Lopes, T. Batista, F. Delicato, and N. Cacho. Composição de eventos para aplicações pervasivas (in Portuguese). In *Proc. of the 26th Brazilian Symposium on Computer Networks (SBRC 2008)*, 2008.
- [19] M. A. Malcher and M. Endler. A context-aware collaborative presentation system for handhelds. In *Proceedings of the 5th Brazilian Symposium of Collaborative Systems (SBSC 2008)*, pages 1–11, 2008.
- [20] F. N. Nascimento, V. Sacramento, G. Baptista, H. K. Rubinsztein, and M. Endler. Development and evaluation of a positioning service based in IEEE 802.11 (in Portuguese). In *Proc. of the XXIV Brazilian Symposium on Computer Networks (SBRC 2006)*, 2006.
- [21] R. Rocha and M. Endler. Domain-based context management for dynamic and evolutionary environments. In *MDS '07: Proc. of the 4th on Middleware Doctoral Symposium*, pages 1–6, 2007.
- [22] G.-C. Roman, G. P. Picco, and A. L. Murphy. Software engineering for mobility: a roadmap. In *ICSE '00: Proc. of the Conf. on The Future of Software Engineering*, pages 241–258, 2000.
- [23] M. Román, C. Hess, R. Cerqueira, A. Ranganathan, R. H. Campbell, and K. Nahrstedt. A middleware infrastructure for active spaces. *IEEE Pervasive Computing*, 1(4):74–83, 2002.
- [24] H. K. Rubinsztein, M. Endler, V. Sacramento, K. Gonçalves, and F. N. Nascimento. Support for context-aware collaboration. *First Intl. Workshop on Mobility Aware Technologies and Applications (MATA 2004)*, 5(10):34–47, 2004.
- [25] V. Sacramento, M. Endler, and F. Nascimento. A privacy service for context-aware mobile computing. In *Proc. of the First IEEE/CreatNet Intl. Conf. on Security and Privacy for Emerging Areas in Communication Networks (SecureComm '01)*, pages 182–193, September 2005.
- [26] V. Sacramento, M. Endler, H. K. Rubinsztein, L. S. Lima, K. Gonçalves, F. N. Nascimento, and G. A. Bueno. MoCA: A middleware for developing collaborative applications for mobile users. *IEEE Distributed Systems Online*, 5(10), 2004.
- [27] D. Salber, A. K. Dey, and G. D. Abowd. The context toolkit: aiding the development of context-enabled applications. In *CHI '99: Proc. of the SIGCHI Conf. on Human Factors in Computing Systems*, pages 434–441, 1999.
- [28] B. N. Schilit, N. I. Adams, and R. Want. Context-aware computing applications. *5th Workshop on Mobile and Ubiquitous Information Access*, Dezembro 1994.
- [29] T. Strang and C. Linnhoff-Popien. A context modeling survey. In *Workshop on Advanced Context Modelling, Reasoning and Management associated with the Sixth Intl. Conf. on Ubiquitous Computing (UbiComp 2004)*, Nottingham/England, Setembro 2004.
- [30] J. Viterbo, M. Endler, and V. Sacramento. Discovering services with restricted location scope in ubiquitous environments. In *Proc. of the 5th Intl. Workshop on Middleware for Pervasive and Ad-hoc Computing (MPAC '07)*, pages 55–60, 2007.
- [31] J. Viterbo, M. Malcher, and M. Endler. Supporting the development of context-aware agent-based systems for mobile networks. In *Proc. of the 2008 ACM Symposium on Applied Computing (SAC '08)*, pages 1872–1873, 2008.
- [32] S. Yau, F. Karim, Y. Wang, B. Wang, and S. Gupta. Reconfigurable context-sensitive middleware for pervasive computing. *IEEE Pervasive Computing*, 1(3):33–40, 2002.
- [33] J. Ye, L. Coyle, S. Dobson, and P. Nixon. Ontology-based models in pervasive computing systems. *Knowledge Engineering Review*, 22(4):315–347, 2007.