# A Middleware Supporting Adaptive and Location-aware Mobile Collaboration

Marcelo Malcher, Juliana Aquino, Hubert Fonseca, Lincoln David, Allan Valeriano, Markus Endler
Laboratory for Advanced Collaboration (LAC)
Pontifícia Universidade Católica of Rio de Janeiro, Brazil
{marcelom, jaquino, hfonseca, lnsilva, avaleriano, endler}@inf.puc-rio.br

## ABSTRACT

*Mobile Applications with location awareness allow mobile users to communicate and share different sorts of location-based information among themselves, such as the current position of other users or geo-referenced data. Although many of such mobile collaboration applications potentially share a good amount of functionality, most of them are developed from scratch, are monolithic and are tailored to specific mobile platforms,, which limit their applicability. This paper presents a client middleware architecture which supports dynamic deployment and composition of components for context- and location awareness, and common collaboration services. We also present some prototype map-based and location-aware applications which we have implemented on the top of our middleware services.*

## Keywords

Mobile Computing, Context-awareness, Location-based Services, Middleware, Dynamic Adaptation, Collaboration, Android.

## 1. INTRODUCTION

Mobile and location-aware collaboration allows geographically distributed and mobile users to communicate and share different sorts of location-based information among themselves, such as geo-referenced annotations or other user's current positions. Although many of such applications potentially share a good amount of functionality related to context-/location-awareness, communication and sharing mechanisms, most of them are developed from scratch, are monolithic and include platform dependent-code , which limits their applicability. The use of middleware platforms is a means for applications to become less dependent of the specificities of mobile platforms and device resources/sensors, and become less complex due to the adoption of high-level software structuring and management mechanisms, as well as reuse of common modules. Hence, major considerations driving the development of middleware platforms for mobile collaboration are built-in support for flexible deployment and composition of services, context-awareness, flexible and asynchronous service interaction model, distribution transparency for data sharing and event distribution, and provisioning of map-based and location-aware services.

During the last two years[1] we have been working on a client middleware architecture that supports the development of mobile and location-aware collaborative applications which features: (i) capability of dynamic deployment of new components and switching between executing components, both at the middleware and the application-layer (ii) extensible context-awareness, (iii) uniform interface for sharing of data and asynchronous communication (among local and remote components), and (iv) support for the combination of basic collaboration services. As a concrete result of this project we designed and implemented basic middleware services that give support for the above mentioned capabilities and a few prototypes of mobile collaboration application clients for the Android platform [1].

This paper is organized as follows. Section 2 summarizes related work on middleware for mobile collaboration. Section 3 shows the proposed client architecture and summarizes its main elements. Then, Section 4 describes in more detail the main middleware elements that we have implemented. In Section 5 we summarize some of the location-based collaboration prototypes that we developed so far, and in section 6 we present our concluding remarks.

## 2. RELATED WORK

Some work have also built middleware for mobile computing with location-awareness, but none of them simultaneously features the capability of dynamic deployment and replacement of components, context-awareness, uniform local and remote asynchronous interactions, and support for composition of common collaboration services. Nevertheless, following two systems share some similarities with our work.

The ContextPhone [2] supports the development of context-aware applications for smart phones. It is composed of interconnected modules which provide a set of open-source libraries and components to be executed on mobile phones. The main modules are the sensor module, which acquires raw context data from different sources,

[1] Work done in the scope of the Mobilis project, a cooperation with TU Dresden, Germany, and the Federal University of Minas Gerais, Brazil; BMBF/CNPq Grant nr. 490817/2006-8

like positioning information from a GPS sensor, and the communication module, which implements connectivity and communication with remote services through different protocols, such as GPRS, Bluetooth, SMS and MMS. However, this platform only provides very simple forms of context-awareness and lacks support for dynamic adaptation of the middleware and applications.

Preuveneers and Barbers [3] describe a resource-aware and context-driven middleware for mobile devices, which is component based and self-adaptive. It follows a layered approach where a run-time layer is responsible for module, component and connectivity management and adaptation control, whereas the context layer is responsible for context retrieval, storage and manipulation of context data. Using their middleware they have implemented a conferencing client comprised of a multimedia, a Jabber and a Web server components and evaluated the energy consumption with and without dynamic adaptation. Despite sharing similar goals and architectural elements as ours, this work is less focused on collaboration components and also lacks a unified interface for local/remote asynchronous communication.

## 3. PROPOSED CLIENT ARCHITECTURE

Our client architecture is composed of an application layer and a middleware layer, as shown in Figure 1. In our approach, a mobile client application (for location/context-aware collaboration) is built out of generic, sharable and dynamically composable components (represented as **Comp_X** in Figure1), each of which implementing an elementary communication or data sharing functionality (e.g. instant messaging, a map annotation service, etc.). The composition, execution, dynamic adaptation and interaction of these components is supported by the following five basic middleware services.

**Component Manager**: is responsible for discovery, dynamic deployment and binding of components used by applications. It also supports queries about all components currently deployed at the device, and their current states (e.g. loaded, deployed, active/inactive).

**Adaptation Manager:** is responsible for triggering dynamic adaptations regarding components of the applications, whenever required. For this purpose, it listens to notifications of context changes, monitors the current configuration of components and requests basic operations on components through the Component Manager.

**Context Manager:** supports the discovery, deployment and execution of any number of Context Providers, each of which collects, processes or distributes context data (e.g. resource states or events, and location or sensor data) from the device's mobile platform.

**Shared Data Manager**: provides a uniform API for asynchronous communication among any component based on a Publish-Subscribe mechanism. A single parameter at the publication operation determines if matchmaking with subscriptions and notifications will happen only to local subscribers, or also remote ones. For the latter, SDM relies on MD-ECI.

**MD-ECI**: is a SIP-based Publish-Subscribe system [4] that supports remote distribution of notifications of publications – which may be data objects or events - among mobile devices. The main difference between an event and a data object is that the latter is kept in persistent storage at the MD-ECI broker for access by late-joiners, i.e. subscriptions made after the object's publication.

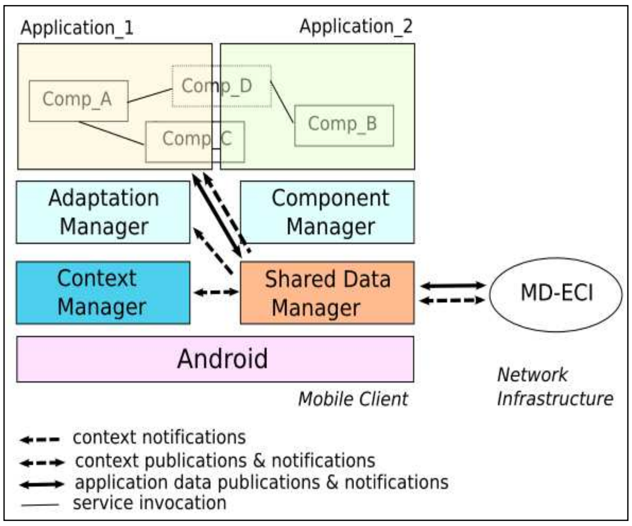


**Figure 1: Client Architecture and MD-ECI**

We chose Android as the primary mobile platform target for our middleware, because it supports Java programming (execution is on the Dalvik VM), defines a Service Oriented Architecture and provides many powerful APIs and libraries for location-awareness, GUI development and access to Google Maps. The Android programming model [1] defines four essential types of elements that make up a mobile application: services, activities, broadcast receivers and content providers. Although most of our current middleware implementation depends on Android features, all its constituent services can, in principle, be ported to any other service oriented platform, such as OSGi..

## 4. IMPLEMENTED MIDDLEWARE SERVICES

In this section we will discuss in more detail the basic services of our middleware. In addition, we also developed some application-level, generic components for mobile communication and location-aware collaboration: a component for (parameterized) proximity detection of mobile devices; a component for sharing geo-referenced data objects, and a component for connectivity-aware instant communication which is capable of buffering outbound messages when it detects a disconnection, and automatically switching from the SDM to SMS communication mode.

### 4.1 Kaluana

Kaluana [5] is the tier of our middleware that implements the Component and Adaptation Managers. It defines a component model on the top of Android's service-orientation framework. In this model, each component defines a set of provided services, a set of used services, and the names of other components it depends on. Also, any Android service or activity is component-based, i.e. has a descriptor which defines the set of services it requires for execution. When an activity or service is started, the **Component Manager** uses the Android Framework to search for the required services. If it finds a locally loaded component that implements this service, it simply activates the component. Otherwise, it may download and deploy a suitable component from a remote component repository.

The **Adaptation Manager** is responsible for determining if a component should be added to, activated, deactivated of replaced from the device, and for selecting the candidate components for such adaptation. This selection is based on the current system context (or user location) and according to an *execution pre-requisite* associated with each component. For example, when the device switches from a GRPS to a WiFi connection (of a specific and trusted SSID), a component for WiFi RSSI-based (indoor) and site-specific location service may be deployed and activated at the device. In order to actually perform such a dynamic adaptation, the Adaptation Manager issues basic activation and binding requests to the Component Manager, which does the activations and re-bindings and then updates its registry.

### 4.2 Shared Data Manager

The Shared Data Manager, or SDM, is an Android service which implements a publish-subscribe mechanism which is used by application and middleware services alike to exchange data and events locally, i.e. for asynchronous interaction with components deployed on the same device, or remotely, with components and applications executing on remote devices.

SDM can be used for sharing almost any type of data. For publishing a data or event, an application middleware service only needs to inform the data/event's subject. Optionally, it may inform other properties (attributes), which are used in subscription expressions for filtering. In case of a data publication, it should add a serialized object representing the data itself, e.g. a geo-referenced text, video-clip or audio recording. To receive updates on a specific subject, an application or middleware component must subscribe with the SDM, registering a listener and optionally informing also an expression referring to the data/event properties. Whenever a new publication on this subject happens, the SDM will notify all subscribers of this subject whose expressions match the properties of the published object/event. In order to deliver data/events to subscribers on other devices, SDM implements also a MD-ECI client.

### 4.3 Context Management Service

The Context Management Service, or CMS, is an Android service that manages the gathering, processing and distribution of any type of context data. Within CMS, each type of context data/event is *de facto* obtained or produced by a specific **Context Provider**. Each of such Context Provider (CP) is a component that the CMS can deploy and activate/deactivate independently, depending on whether there is some application component interested in the corresponding context type. CMS also supports the discovery and dynamic download of new Context Providers from a remote Repository of CPs.

CMS uses SDM to deliver the requested context data object to the subscribers, regardless if they are local or remote. Context subscribers may be application specific components or other Context Providers, such as those responsible for transforming or aggregating lower-level context data and producing high-level context information. CMS also provides class ContextConsumer, aimed at hiding from the application the code necessary to interact with SDM and CMS, and thus offering a much simpler interface, referring only to the specific context information needed.

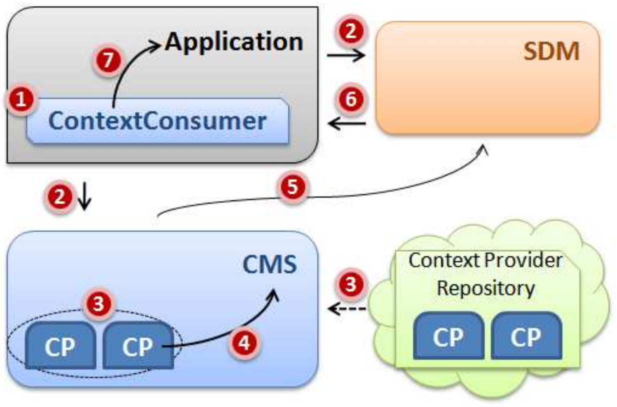


**Figure 2: CMS interactions upon new context subscription.**

Figure 2 illustrates the basic interactions between an application, the CMS and SDM: A ContextConsumer of an application issues a subscription at SDM for some specific type of context information, e.g. Battery level, and notifies CMS (steps 1+2); Alternatively, the ContextConsumer may also make a synchronous request to obtain the current state of a specific context type (step 2). In either case, CMS searches for any locally deployed Context Provider that can produce the requested context information. If none can be found locally, CMS searches, downloads the corresponding CP from the remote repository and activates it (step 3). Once activated, the CP polls or invokes methods at the Android API of the corresponding Resource Manager, and delivers the polled data to CMS as a ContextInformationObject (step 4). CMS adds some data to this object (e.g. the deviceID and timestamp), and publishes it through SDM (step 5). This object is then delivered to all ContextConsumers with matching subscriptions (step 6), which in turn pass it to the applications for specific handling (step 7). Currently, we are setting up a library of Context Providers for CMS that includes following types of context: Geographic location (GPS), Time, Battery level, Type of wireless connection, WiFi RF signal strength, Accelerometer, etc.

## 5. LOCATION-BASED COLLABORATION APPLICATIONS

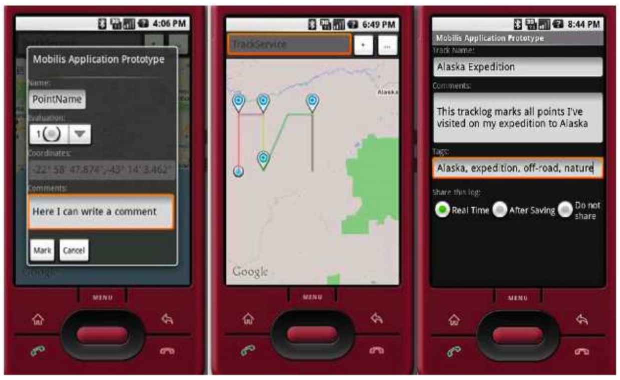
Using our middleware and application-level components, we implemented some prototype applications for location-based mobile collaboration, some of which are as follows:

**Geo-tagging** is an application similar to Google's Latitude [6] which allows mobile users to visualize (on the digital map) its own and other user's position in real time while they move, and to create - and also instantly share with other users - geo-referenced Points of Interest (POIs), or tags, e.g. for city landmarks or attractions. Geo-tagging uses a component for sharing geo-referenced objects and events, which in turn uses SDM and MD-ECI to remotely distribute user and tag attributes associated with GPS coordinates retrieved by the Context Provider GPSLocationCP within CMS. But, whenever the user's device is unable to retrieve its current GPS position, this will be detected by the PositioningAvailabilityCP in CMS, and will cause the Adaptation Manager to de-activate the application component of Geo-tagging responsible for the creation of new tags .

**TrackService** is used for creating Off-road/Trekking routes/tracks, grading track sections, and sharing this information (i.e. the track log) with other mobile users. The mobile user is able to log his/her route as a sequence of GPS coordinates of his/her mobile device. Figure 3 shows three possible screens of TrackService. While the user is walking/driving along a route, he/she is able to evaluate the route according to the difficulty or danger level, giving a grade from 1 to 10 for each route section (cf. Figure's leftmost screen). As he/she gives grades to the sections of the track, a different color is used to represent each difficulty/danger level, ranging from green (e.g. easy) to yellow, and then to purple (e.g. very difficult). The user can also choose if, he/she wants to share his track with others (cf. rightmost screen), and the options are: in real-time, after saving it on the device, or no sharing..

Other prototypes developed include: **GeoCast**, an application where messages can be sent to any registered mobile user whenever he/she enters or leaves an arbitrary geographic region (marked as a polygon on the map), or gets closer, or farther away, from a specific point marked on the map;

**Bus4Blinds** is an application that notifies a person with serious visual impairment waiting at a bus stop when a bus of a selected line is approaching the bus stop where the user is waiting. The notification is symmetric, meaning the bus driver is also notified of the presence of the bind person at the stop.


Mobilis Application Prototype
Name: PointName
Evaluation: 1
Coordinates: -22° 58' 47.874", -43° 14' 3.462"
Comments: Here I can write a comment

Mobilis Application Prototype
Track Name: Alaska Expedition
Comments: This tracklog marks all points I've visited on my expedition to Alaska
Tags: Alaska, expedition, off-road, nature
Share this log: [✓ Real Time] After Saving Do not share

**Figure 3: Screenshots of TrackService**

## 6. CONCLUSIONS

So far, our effort was mainly focused on the development of basic middleware mechanisms and services, such as Kaluana, SDM and CMS. However, there are still many things to be improved, specially with regard to decision making, selection of components and safe execution of dynamic adaptations by the Adaptation/Component Managers of Kaluana. We are also aware of the scalability problems of our remote context distribution approach and aim at developing a context sharing architecture based on federated MD-ECI brokers. In parallel, we plan also develop other generic, shareable components to be used by such mobile collaboration applications.

## 7. REFERENCES

[1] Google Android (2009) from http://www.android.com/
[2] Raento, M., Oulasvirta, A., Petit, R., & Toivonen, H. (2005). ContextPhone: A prototyping platform for context-aware mobile applications. IEEE Pervasive Computing, 4(2), 51-59.
[3] Preuveneers, D., & Berbers, Y. (2007). *Towards context-aware and resource-driven self-adaptation for mobile handheld applications.* Proceedings of the 2007 ACM symposium on Applied computing.
[4] MD-ECI (2009) from http://www.lac.inf.puc-rio.br/moca/mdeci/mdeci.htm
[5] Fonseca, H. (2009) A Component-based middleware for Dynamic Adaptation on the Android Platform, M.Sc. Thesis, Depratment of Informatics, PUC-Rio, Brazil.
[6] Google (2009). Latitude Apps , from http://www.google.com/latitude/app