# Chapter 45
# Creating Applications for Real-Time Collaboration with XMPP and Android on Mobile Devices

**Daniel Schuster**
*TU Dresden, Germany*

**Dirk Hering**
*TU Dresden, Germany*

**István Koren**
*TU Dresden, Germany*

**Benjamin Söllner**
*TU Dresden, Germany*

**Thomas Springer**
*TU Dresden, Germany*

**Markus Endler**
*Pontifical Catholic University of Rio de Janeiro, Brazil*

**Alexander Schill**
*TU Dresden, Germany*

## ABSTRACT

*The goal of this chapter is to discuss the challenges of generic protocols and platforms for mobile collaboration in general and for the adoption of XMPP for mobile collaboration in particular. The chapter will introduce the XMPP protocol family, discuss its potentials and issues for mobile collaboration, and describe experiences with the implementation of mobile collaborative middleware and applications based on XMPP. In particular the protocol family has been used to create a generic middleware for mobile collaboration providing a set of generic services such as publish/subscribe, group management, and chat functionality, as well as advanced functionality for geo-location and geo-tagging, map visualization, and multimedia content sharing. For the implementation of our platform and applications XMPP is used in combination with the Android platform running on the mobile devices. The authors describe their experiences in adjusting and adopting XMPP protocol implementations based on Java on the Android platform.*

DOI: 10.4018/978-1-61520-655-1.ch045

## INTRODUCTION

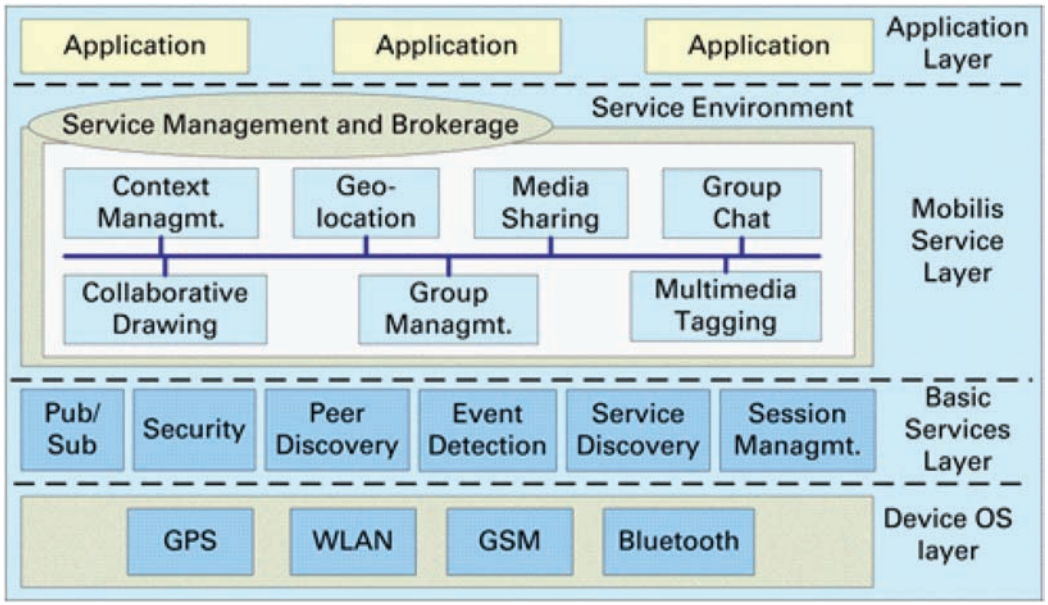
There is already a multitude of collaborative applications available in mobile environments. Although they share a good amount of common functionality, most of them are built from scratch, or are tailored to a specific device platform using proprietary libraries. An open and customizable environment for mobile collaborative applications is still missing. To set up a generic environment for mobile collaboration support, the selection of the right set of underlying protocols is of high importance. Based on the protocols for collaboration the foundations for interoperability, scalability, portability and performance are created.

In earlier work (Springer et al., 2008) we introduced the Mobilis reference architecture as a service-oriented approach to support developers of mobile collaborative applications with a framework covering all the different aspects and layers of such applications. This comprises the device operating system, basic communication and context services, a service environment with commonly used functionality as well as the applications at the application layer. As can be seen in Figure 1, each of these four layers comprises a distinguished set of individual services. The functionality of the services at the Mobilis service layer will be described later in this chapter.

The Mobilis reference architecture already provides a good guideline for developers of mobile collaborative applications. It served already as a basis for the development of a set of applications adopting diverse collaboration functionality:

- **MobilisFunFlags:** So-called fun flags can be tagged to locations at a map to notify other users of the application about cool places. Images and text notes can be attached to these places.
- **MobilisGuide:** Tourists travelling together can create and join closed groups and are able to be aware of the other group members visualized as icons on a map-based view. They can interact by selecting icons of group members on the map, for instance to chat with each other or contact one another directly.

*Figure 1. Mobilis reference architecture*



- **MobilisBuddy:** Users get an alert if friends from social networks such as Facebook are in proximity (for instance if the distance is less than 100 meters) and can locate them on a map or call them.
- **MobilisTrader:** Users of this application offer products such as a used washing machine or mobile phone together with the location of the product. Other users may insert requests for products. If request and offer matches within a pre-defined distance, both users get an alert and may contact each other.

These activities have been mainly carried out to drive the identification of appropriate services at the Mobilis service level which can be reused in different applications. The implementation of the various applications also enriched our experiences with the service platform. During the prototype development we also tested different target platforms. As described in (Springer et al., 2008), we implemented the MobilisGuide application for the three different platforms Java ME, Java SE, and Android. Furthermore, we tried different alternatives to realize the basic connectivity encapsulated in the Basic Services layer such as NaradaBrokering (Pallickara & Fox, 2003), Google App Engine (Google, 2009), and the eXtensible Messaging and Presence Protocol (XMPP) (XMPP, 2009).

While all these approaches remain compatible to the Mobilis reference architecture, each of them provides insights into different facets of the chances and challenges of development of mobile real-time collaboration applications. In this chapter we focus on the experiences made with XMPP on the Android platform as the most appropriate approach for the realization of systems according to the Mobilis reference architecture. We consider XMPP as one of the key technologies because it is a standardized, open, and widely adopted protocol for collaborative applications in the internet. It provides a large set of mechanisms for key collaboration features like group management, multi-user chatting, publish/subscribe, and multi-media streaming which is easily extensible by specifying XMPP Protocol Extensions (XEPs). Moreover, Java-based implementations are available and are ready to use. In combination with Android, key features of mobile collaborative applications are easy to implement. Such features comprise map-based awareness, exchange of location and presence information, and content management.

In the following, we introduce the XMPP platform as an XML-based infrastructure for mobile collaboration and the Android technology. While the choice of Android does not influence the conceptual architecture for XMPP-based mobile collaboration presented in the Concepts and Solutions section, it introduces some specific issues at implementation level which are discussed afterwards. In this section, we will show more detailed concepts, XMPP-based protocols, and implementation issues of two selected applications (MobilisGuide and MobilisBuddy) out of the applications mentioned above.

But before we get into details about all these issues we want to motivate our work with a discussion about the current challenges of mobile real-time collaboration in the following section.

## BACKGROUND

### Major Challenges for Protocols and Middleware for Mobile Collaboration

Communication and collaboration are major driving forces for people to use today's network infrastructures. Over the last decade, a number of new communication and collaboration technologies, such as audio and video conferencing, VoIP, Instant Messaging, Blogs, Wikis, application sharing, shared editing tools, etc. have emerged and created a heterogeneous bundle of interaction channels to the final user.

Consequently, this "escalating variety of communication devices and the ever increasing volume of messaging activity" (Hutton, 2001) has greatly increased the communication complexity for both the initiator and the recipient of a communication request, e.g., initiators have to think about the recipient's location and context and the appropriate communication channel while recipients are confronted by a myriad of communication devices, addresses and services. This creates a fragmented communication setting whose coordination is time consuming and error prone. Moreover, due to increased flexibility, spontaneity and asymmetry of interactions people are potentially confronted with a level of interaction that might exceed their personal preferences, causing what Sørensen calls the "interaction overload" (Sorensen et al., 2002). In order to cope with these problems, future communication and collaboration middleware will strive towards unified communication technology and powerful filtering mechanisms.

The confluence of several trends in the design of portable devices, cloud computing services, sensor technologies, geospatial information systems, and wireless networks, as well as the emergence of new forms of end-user interactions and visualization technologies are leveraging the technological base for advanced real-time collaboration in mobile environments. Anyway, because of the specific characteristics of mobile devices, wireless network technologies and the specific requirements of mobile users, applications have to be tailored to the special characteristics. As a result, applications for real-time collaboration on mobile devices face the following challenges:

**Enriched Presence:** An important aspect for mobile collaboration is to increase the awareness of collaborating parties about where the others currently are, what they work on and how what social context they are within. Thus, the provisioning of enriched presence information as well as intuitive means for users to manage preferences of its disclosure is of high importance. The idea of presence information is to signal to the initiator of a communication action the recipient's "ability or willingness to communicate" (de Poot et al., 2005). In current IM tools presence information is either manually provided by the user or deduced by technical means, e.g., the user is not logged into the system, or the device not connected. However, to support interactions with mobile users it may be useful to include also automatically sensed or inferred presence information about the user's location or context, e.g., "inside car in motion", "in a movie hall", "in the boss's office" etc., as this may also reveal the recipient's availability to initiate or continue a communication session. Thus, if this enriched presence information is shared with the mobile user's interaction partners, they may also become aware of the current interaction restrictions of the mobile user, and may lower their expectations of immediate reactions or responses. In order to implement such enriched presence information, future collaboration systems will probably need to be integrated with middleware for sensing, context inference and distribution.

**Unified communication:** According to (Hutton, 2001), unified communication can be defined as the integration of different synchronous and asynchronous communication and collaboration modes - preferably based on an unique and intuitive metaphor - and aimed at reducing the fragmentation and complexity of today's interactions. On the other hand, powerful filtering shall enable: (i) flexible selection of time periods, the persons and the channels at which the user is willing to interact, and (ii) diverting or transferring incoming communication requests between channels and devices according to a set of user-provided or automatically learned filters or rules. Such rules might relate time ("at office hours"), situations ("at a meeting", "on travel"), callers ("co-worker", "chief") and interaction modes ("voice call", "instant messaging").

**Usability and Quality of Experience:** Features like application sharing or shared editing of documents and other multimedia content are well established in desktop environments. In mobile

settings, these features are desirable and helpful as well. Due to the limitations of mobile devices and the restricted bandwidth of wireless network technologies, the propagation of local changes to all collaboration partners is quite challenging. Anyway, the user expects the same quality of experience as provided by desktop applications. Beyond the real-time aspect of collaboration, the interaction of users with applications on mobile devices requires interaction types and user interfaces tailored to the characteristics of the used devices.

**Web 2.0 Integration:** Social contacts, multimedia content and already established infrastructures for collaboration in the Web 2.0 are important resources for mobile collaboration. Social networking gains increased popularity and is the base of information about the contacts and relations of users and thus provides extended information about potential collaboration partners. Moreover, platforms for sharing media content like Picasa, Flickr or YouTube contain multimedia data which can potentially serve as the object partners collaborate on. An integration of these resources provides users a seamless handover between familiar web-based environments and mobile applications and prevents the parallel maintenance of social contacts and content.

**Configuration and Adaptation:** Due to the heterogeneity of mobile devices and wireless network technologies a "one-size-fits-all" application would fail to meet the challenges raised by heterogeneity. Applications and the underlying platform should be highly configurable to the capabilities of particular mobile devices and the available connectivity. This comprises the consideration of particular user needs and preferences. To cope with dynamic changes of resource availability at runtime, application functionality and data should be adaptable taking the context into account.

**Security and Privacy:** Another important challenge is raised by the need for exploiting the location information and extended awareness features. At the one hand these features can significantly improve the interaction of collaborating parties. At the other hand, disclosure of private information is required which could raise major privacy concerns which could prevent users from adopting the developed systems. Therefore, a balance between awareness and privacy has to be established. This implies especially to support the user to keep control what and with whom his or her private information is shared. While this issue is out of scope of this chapter, the authors refer to other research dealing with these aspects (see Groba et al., 2007 and Franz et al., 2008).

## Overview of XMPP

The Extensible Messaging and Presence Protocol (XMPP) (XMPP, 2009) is a family of protocols for internet-based collaborative environments and is standardized by the Internet Engineering Task force (IETF). It adopts a client/server approach with multiple interconnected servers (like e-mail) and is based on XML. A set of XEPs (XMPP Extension Protocols) provides further functionality, e.g., presence and location exchange, publish/subscribe, file transfer, group management and multi-user chat. These extensions are driven and managed by the XMPP Standards Foundation (XSF), an open, non-profit organization as well as a world-wide community of developers. Having its roots in instant messaging and presence, the XSF widened its scope towards supporting real-time communication and collaboration on top of XMPP. Anyway, to use these protocols for mobile settings, they have to be adjusted to resource constraints, limited connectivity, and varying situations of use. Features for location and context-awareness should be integrated as well as adaptation concepts, new interaction forms and collaboration services to create collaborative applications which are really valuable for mobile users.

## XML Stanzas

The actual information exchange in XMPP runs on top of a TCP connection and inside an XML stream, which is started for both directions at the beginning of the communication, right after the initial mechanisms for authentication and encryption. The specific semantic transmission units are called stanzas; thereby each of them is a well-formed piece of XML. There are three types of stanzas which we will depict in the following: Message, Presence and Info/Query.

### Message

The Message stanza can be regarded as a push mechanism, whereby a user may send information to another entity on the XMPP network. In Instant Messaging scenarios, a message is typically encapsulating chat data and therefore features a body part similar to an e-mail. Moreover, messages are used for event delivery and notifications. Therefore, the attached data is embedded into an event tag with a custom namespace. For instance, the publish/subscribe extensions heavily rely on the capabilities of this stanza protocol as will be shown later.

```
<message from='ariel@island.lit/cell'
to='prospero@island.lit'
id='msg1'>
  <thread>Shakespeare's Play</thread>
  <subject>The Tempest</subject>
  <body>
    That's my noble master!
    What shall I do? Say what? What
shall I do?
  </body>
</message>
```

### Presence

Presence data shortly includes all information concerning the availability of a user, whether 'online', 'not available' or different, arbitrary values. The presence mechanisms can be seen as a simple broadcast system, through which the presence information can be published to subscribed users. The subscription itself is maintained through entries of the roster, which is the contact list of the particular identity. Thereby, upon each update, the XMPP server propagates the Presence stanza to all users that have this particular member in their roster.

```
<presence from='ariel@island.lit/
cell'to='prospero@island.lit'
type='available'>
  <status>Waitin' for the noble mas-
ter</status>
  <priority>10</priority>
  <show>chat</show>
</presence>
```

### Info/Query

The Info/Query (IQ) protocol features a stanza model that is employed for request/response mechanisms that involve another entity. The four possible types are get, set, result and error. Get and set requests have to be answered with either a result or an error stanza with the appropriate identifier attribute. IQ interactions follow a common pattern of structured data exchange; therefore, payload is attached as a child element. The protocol is used by the service discovery means of XMPP; furthermore we heavily apply it in the realization of our conceptual architecture with XMPP. Queries are accomplished by using the <query> tag.

```
<iq type='get' from='ariel@island.lit/
cell' to='plays.shakespeare.lit'
id='info1'>
  <query xmlns='http://jabber.org/
protocol/disco#info'/>
</iq>
```

## Extensibility

The XMPP protocol is designed to be extensible by means of the existing stanza models. Therefore, all enhancements are based on the three building blocks that were mentioned before. Extensions are published as XMPP Extension Protocols (XEP) and run through a community-driven standardization process that starts with an experimental version, progresses as a working draft and is closed by providing the final standard. Currently, numerous XEPs exist whereof three extensions are briefly introduced which are important for our work.

### Multi-User Chat

The Multi-User Chat XEP-0045 (Saint-Andre, 2008) describes a protocol for multi-user text chats, whereby users can exchange text messages in the context of a room. More general, the MUC can be seen as a means for holding particular members together in the character of a group. MUC features control models that allow defining moderators and administrators to change certain users' rights. Additionally, Multi-User Chat rooms may be secured by a password and also only accessible upon invitation.

### Publish-Subscribe

The Publish-Subscribe extension XEP-0060 (Millard et al., 2008) is based on the publish/subscribe or observer pattern. An interested participant subscribes to a certain node. If an entity publishes information to this node, the interested participants get an update message. The extension provides a service to create the nodes that may be used to publish information to. Just like the Multi-User Chat, the Publish-Subscribe extension provides various content models and user affiliations for controlling the access to the nodes. The possible roles of an entity are 'owner', 'publisher' or 'outcast' (who may not access the node); by default entities may only subscribe for notifications. Nodes may be combined to a collection while a collection is again a node. Information may only be published on leaf nodes.

For our goal of creating mobile applications in a resource-aware way, the publish/subscribe pattern is of high importance.

```
<message from='pubsub.shakespeare.
lit' to='francisco@denmark.lit'
id='foo'>
  <event xmlns='http://jabber.org/
protocol/pubsub#event'>
    <items node='princely_musings'>
      <item id='ae890ac52d0df67ed7cfd
f51b644e901'>
        [ ... ENTRY ... ]
      </item>
    </items>
  </event>
</message>
```

### Service Discovery

The Service Discovery, described in XEP-0030 (Hildebrand et al., 2008) specifies a protocol for discovering services, their properties and features of another XMPP entity. Beneath, supported extension protocols of the entity can be identified. The Service Discovery uses I/Q stanzas for querying the opponent. For example, the MUC chat extension presented above uses a service discovery stanza for querying the server for a list of available groups. An entity may have more identities

which are exposed by the "identity" tag within the I/Q result. The "feature" tag is used to describe the features; it contains an attribute called "var" whose value is a protocol namespace.

## XMPP for Real-Time Collaboration

These widely available extensions as well as a few other characteristics of XMPP make it well suitable for real-time collaboration applications. To underline this claim, we want to outline some of the rationales for choosing XMPP as an infrastructure for our mobile collaboration framework.

### Session and Group Management

XMPP can easily create sessions between users of collaboration applications and ease group management. XMPP already provides mechanisms to login to a server and contact other users by a world-wide unique ID (the JID). Furthermore it supports the creation of multi-user chat rooms through one of its extensions. Clients joined to a chat room are able to see and contact each other. Access restrictions can be made on group membership.

### Community Support

The XMPP Standards Foundation is a vital community that fosters the development of real-time collaboration. Big industry players base their real-time collaboration solutions on XMPP and standardize them as XEPs. XMPP servers as well as XMPP clients and XMPP client libraries are available as free and/or open source software for a number of platforms including Java.

### Asynchronous Messages

As XMPP offers long-living client-server connections, we can use XMPP to send asynchronous server-client messages. Unlike traditional client-server systems we have a need to establish a bidirectional link to enable the server to send messages asynchronously to the client in real-time collaboration. Using XMPP for this purpose is very easy as we can directly send messages to any JID.

### Extensibility

XMPP offers great ways of extensibility as all info/query messages are defined within their distinct XML namespace. The XML namespace concept can be used to easily define new protocols to be transported within info/query messages. An XMPP extension for service discovery eases the deployment of new extensions.

### Publish/Subscribe

One of the most powerful general-purpose services for real-time collaboration is publish/subscribe. It can be used for all kinds of awareness mechanisms such as position updates within a group. XMPP already offers an extension for hierarchical publish/subscribe that can be used for that purpose.

While we rely on XMPP for the protocols between clients and server, mobile real-time collaboration can only be realized as a whole technology stack, taking a closer look at the client platform. In our case we chose the Android platform as it supports easy realization of map-based applications, support for XMPP as well as a comprehensive application framework.

## Overview of Android

In short, Android is a software stack for mobile devices including an operating system, a runtime environment and key applications (see Figure 2). The platform was introduced in November 2007 by the Open Handset Alliance (Open Handset Alliance, 2009). The Open Handset Alliance (OHA) is a group of currently 47 technology companies with the objective to "offer consumers a richer, less expensive, and better mobile experience" and proclaiming to "revolutionize the mobile

industry" (Open Handset Alliance, 2009). The main driving force behind the coalition is Google who initiated the cooperation and is responsible for the actual Android software stack. The Open Handset Alliance committed itself to providing "open" solutions for mobile technology; they believe that the open nature of their products whose architecture may be visible to everybody leads to more innovative services and a rich portfolio of easily operated applications.

Android as the current core product of the Open Handset Alliance is basically a stack of a Linux-based operating system and a software platform designed for its extension. The Android environment itself is bound to openness which is characterized by the way of integrating and handling additional software: Any part of the system is replaceable, be it the home application, the unlock screen or the web browser; there are no privileged pieces of software. Thereby, the platform significantly differs from current operating systems for mobile devices such as the iPhone or Symbian. In particular the approach of managed code makes developing software for Android easier, while Symbian developers have to manually consider memory allocation issues.

However, in this chapter the main focus concerning Android is its core libraries and their efficient use. Therefore, in the following, the fundamental concepts of developing applications for Android are discussed.

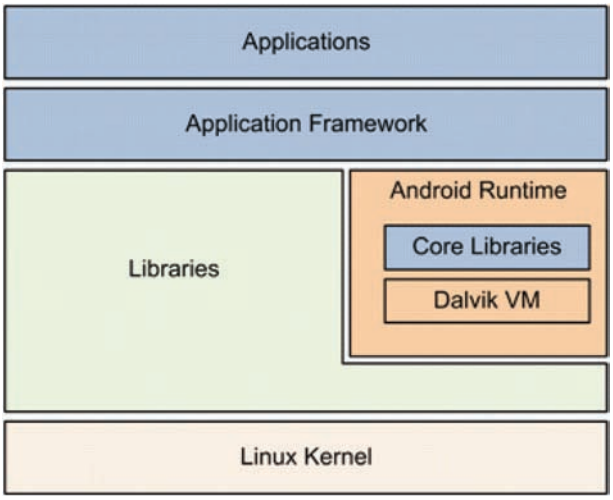
## Android Architecture

The Android platform is highly based on components and characterized by strong modularity. At the bottom, a Linux kernel provides the essential memory management, a network stack and a driver model so it can be regarded as a hardware abstraction layer for the software stacks above, which are discussed below.

### Dalvik Virtual Machine

The Dalvik Virtual Machine is the core part of the software stack to run custom applications. The most noteworthy issue about this runtime component is its architecture and format. While the programming language for developing applications is Java, Google decided explicitly not to use a regularly available Java Virtual Machine but an own implementation. By providing an own implementation, Google was able to release a Virtual Machine which is optimized for mobile environments from scratch and therefore has low memory footprint and minimal requirements. Un-

*Figure 2. Android platform architecture*



like the stack based standard Virtual Machines, Dalvik is register based and by that achieves a greater ahead-of-time optimization. By relying on the underlying platform for address space separation and threading support, Android allows multiple Dalvik instances at once in contrast to other platforms that operate mostly by running in one single process.

Dalvik is not able to run bytecode produced by standard Java compilers. Instead, the format used is the Dalvik Execution Format (dex), which is optimized for efficient storage and memory-mappable execution. However, Java source code files are first translated into Java bytecode by a standard Java compiler, before a cross-compiler converts the results to Dalvik bytecode. Because of this sequence, both Google benefits of a huge developer community and developers are able to profit from existing development environments; solely the cross-compiler has to be run separately. Additionally, many libraries are available for being employed by applications, like the Smack XMPP library which we employed in our prototypes.

### Application Framework

The Android platform is optimized for simple reuse of components. Therefore the Application Framework builds an abstraction layer, where applications can both publish their capabilities and access the operations of other modules. Thereby, the framework is responsible for enforcing security constraints. For being able to access certain resources, permissions have to be requested by the application that can be granted by the user upon installation. The Android platform is also already provided with numerous default permissions; for instance, network connection is granted to components that applied for the "network" permission. Additionally, for new functionalities, permissions can be easily introduced to the system by the means of namespaces.

Beyond, the Application Framework features a set of APIs, for instance those responsible for telephony, location awareness and the view system. Finally, the Resource Manager enables applications to access non-code resources in a flexible way. Thereby, providing localizations and custom views for different screen orientations is easily affordable.

## Android Applications

An application for Android runs on top of the software stack and is interpreted by the Dalvik Virtual Machine. It is distributed in the Android Package format (apk) which contains all the code and its resources like images and localization files. Every package may combine four building blocks, which will be depicted in the following. Finally, all elements, the data they are able to handle and the content published to other components are declared in an XML-based manifest file.

### Activity

Basically, an Activity can be thought of as a screen of an application into which the particular user interface views are embedded. Two concepts exist for navigating between screens. First, an activity may start another activity without any further dependence. Second, the started activity may explicitly be defined as sub-activity, so that a return value is provided for using it in the asking activity by a predefined course of action.

The actual layout of the comprised views may be specified in the code like in common view libraries for desktop applications such as AWT or Swing. Additionally, a more modularized approach is applicable, where the layout may be inflated by declaring it in a proprietary XML format accessible by the Resource Manager.

### Intents

The Intent mechanism is one of the core characteristics of the Android platform (Android, 2009). Generally, an Intent describes what an application

wants to be done by another part of the system and therefore can be generalized as a type of Remote Procedure Call. The data structure inherited in form of a Java object consists of first the Action string acting as an identifier, second the optional data represented by a Uniform Resource Identifier (URI) and third additional payload attached as an extra, which can be specified in multiple formats including strings, arrays and other serializable objects.

The counterpart of an activity that sends an Intent is the component able to process the request. Therefore, an Intent Filter expressing the ability to handle a certain action has to be supplied. Now, whenever a component requests certain functionality described by the Action, the system searches the appropriate component that has a compatible Intent Filter and delegates the further processing.

Besides, even navigating between screens is done by resolving Intents at runtime. Activities are able to reuse functionality by making Intent requests; for instance, the web browser can be opened at a certain page, by setting the appropriate URI. Additionally, activities can be replaced by publishing an equivalent Intent Filter. Finally, Intents can be used as a broadcast system for informing a range of applications about external events. Therefore, a Broadcast Receiver has to be implemented, not necessarily showing up a user interface.

### Service

A Service can be generalized as a long running background thread, which runs without a user interface. Activities can bind to a service and manipulate it by particular interfaces provided by the service. However, a service stays enabled even after navigating away to a different screen, until it gets stopped.

### Content Provider

By offering a Content Provider, any application is able to make its data accessible to other components; the concept is very much like a database query mechanism and is the only way to share data across packages. Therefore, a certain set of methods for data querying and manipulation is provided. For instance, the Android platform already provides a number of Content Providers for data like the contact list and the available images.

## XMPP on Android

When the Open Handset Alliance released the first version of the Android Software Development Kit (SDK), the Android library featured a Peer-to-Peer API that was using XMPP as transport means. The actual XMPP implementation was accessible by a Service and provided mechanisms for presence and message exchange; the main application area was described to be easy message passing between any two devices. Therefore, a Google account was required.

However, in the 1.0 version of the SDK, the service was removed because of security issues, as remote messages were transformed to a local Intent. Therefore, our implementation uses a custom approach by adopting the Smack XMPP library.

## CONCEPTS AND SOLUTIONS FOR XMPP-BASED MOBILE COLLABORATION

After introducing what XMPP and Android already offer to realize mobile real-time collaboration we now have a closer look at our architecture and applications to add the still missing pieces. Our architecture is a concretion of the Mobilis framework shown in Figure 1.

## Conceptual Architecture

The Mobilis framework was built according to the paradigm of Service-oriented Architectures (SOA). It offers a flexible environment of collaborative services that should ease the development of mobile collaborative applications. Clients can be such devices as smartphones, PDAs, or laptops. We assume a running Internet connection, although infrequent disconnections may occur. A GPS receiver or other localization technique is required to use location-based services.

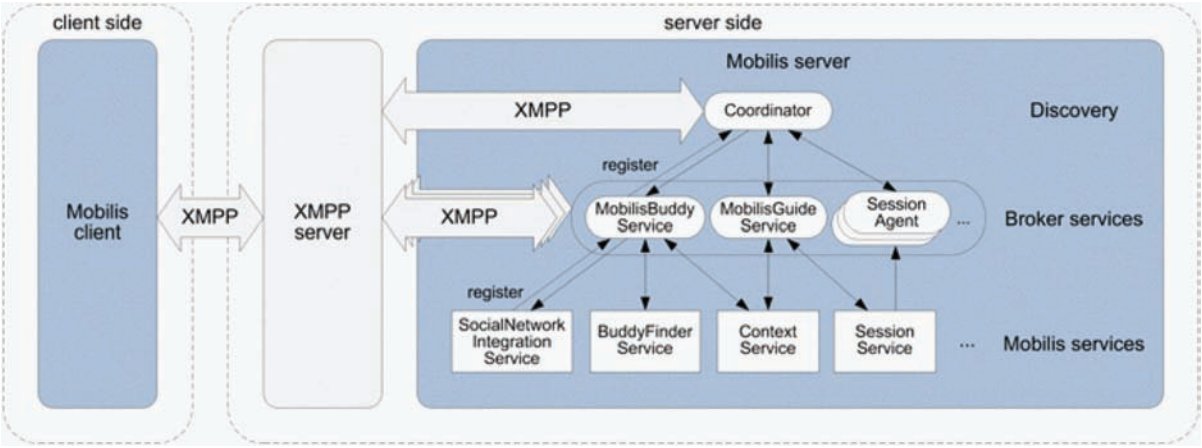
The client communicates with the server using the XMPP protocol. XMPP offers a bidirectional XML stream between client and server and thus enables asynchronous updates to be sent from the server to the client as well as vice versa. The basic protocol elements of XMPP (message, presence, info/query) are used to realize the inter-service protocols of our framework, while especially the info/query primitive is very useful to realize request-response services.

The server side has two distinct components: the XMPP server and the Mobilis server. We assume an existing XMPP server implementation to be used as a black box. No native extension is required as all Mobilis server components run as XMPP clients and are thus fully supported by each available XMPP server implementation on the market. Even a public XMPP server such as jabber.org can be used at the server side while an own process at the same machine as the Mobilis server is preferable regarding the performance of communication. The Mobilis clients as well as the central server components each register themselves with a globally unique Jabber-ID (JID) at the XMPP server. After opening bidirectional XMPP connections to the XMPP server, all of these entities are able to send XMPP stanzas to each other.

As can be seen in Figure 3, we have three layers at the server side: the Discovery at the top with the Broker services layer below, and the Mobilis services. While the latter can already be found in the reference architecture in Figure 1, the broker services and discovery layer realize the logical component "Service Management and Brokerage". The basic service layer is fully replaced by XMPP and its extensions (XEPs), which can be viewed as individual basic services integrated into XMPP.

The Coordinator is a singleton instance known by the clients and is contacted by each collaborative application at its first attempt to use the Mobilis services. It redirects the clients to individual Broker services. A Broker service receives

*Figure 3. XMPP-based architecture*



XMPP messages from a collaborative application (e.g., MobilisGuide or MobilisBuddy) and processes these requests. As each Broker service has its own XMPP connection to the XMPP server it can easily be run on a separate machine if this is necessary to ensure scalability.

Each Broker service is a composite service that uses some subordinate Mobilis services for its functionality. As can be seen in Figure 3, Mobilis services register themselves at Broker services while Broker services register at the Coordinator. This staged SOA concept takes care of the different requirements of collaborative applications running inside the Mobilis environment. Some of the Mobilis services such as the MobilisBuddy service are application-specific while others such as the Context Service are used by different Broker services.

## Group Management

Although in the past XMPP-based architectures were mainly used for Instant Messaging, the protocol allows to be employed in a wide variety of application areas, as we have shown above. For example, the "group" metaphor is best corresponded to the Multi User Chat (MUC) capabilities of XEP-0045 (Saint-Andre, 2008). Consequently, in our framework MUC is one of the building blocks of mapping collaboration artifacts to the conventions of the protocol.

The Group Management Service on device level is tasked with handling all group membership related matters which includes being aware of all the existing groups at the server. That is why it is amongst the first services to be used on client side, that is to hand out a list of all available groups for displaying it to the user and letting him or her decide which one to join. To get this directory, the Session Coordinator is asked by sending an IQ stanza with type "get", including a custom query with the namespace "mobilis:iq:groups". Similar to the Service Discovery extension mechanism (Hildebrand et al., 2008), the server returns a list of items each with the name of the group and the Jabber ID of a respective Session Agent. In the context of the server architecture described above, the Session Agent is implemented as a Broker Service and always responsible for one group at a time only.

After the user selected the appropriate group to join, the client needs to send an IQ query with the namespace "mobilis:iq:joingroup" to the Session Coordinator. Explicitly the coordinator has to be contacted, as in cases where the group reached its capacity limit or is no longer able to accept the particular user because of other reasons, the coordinator may suggest alternatives. Now that the coordinator received the request, its task is to delegate it to the proper Session Agent. The Session Agent henceforward knows the Jabber ID of the user claiming for membership and is now able to send out an invitation to the client. The invitation is received by the Group Management Service which hereupon accepts it and registers both the group name and the appropriate agent's JID into the list containing all the group belongings.

If the desired group of the user does not yet exist, or if there are no groups at all yet managed by a Session Agent, the user may create a new group session. Therefore, as described in the Session Coordinator section earlier, an IQ query stanza with the namespace "mobilis:iq:creategroup" has to be sent to the Session Coordinator. Thus triggered, the coordinator initializes a new Session Agent which creates a Multi User Chat room and the associated publish-subscribe nodes. Now the agent is able to send the group invitation which is handled by the mobile client in the same way as the invitation for joining a room.

## Proximity-Based Services

Special attention was paid to build proximity-based services like MobilisBuddy and MobilisTrader. In the following, we outline architecture and protocols of the service underlying MobilisBuddy. As can be seen in the section "Conceptual

Architecture", the MobilisBuddy service is the broker service responsible to provide the social networks-based buddy finder functionality. It uses the following services (see Figure 4):

- **Social Network Integration Service:** Aggregates and combines information from all social networks the user is currently connected to.
- **Context Service:** New context information arrives at this service. The context service receives location updates from the clients and maintains their positions per JID. Other Mobilis services, which are interested in incoming context information may implement the ContextAwareness interface and register at the context service to receive this information. An example where this mechanism is used is the BuddyFinder service.
- **BuddyFinder Service:** This service requests a list of friends from the Social Network Integration Service when it receives an incoming location update. This list consists of friends from connected social networks who are currently also signed on to the Mobilis platform. Afterwards, the BuddyFinder service executes a bi-directional **proximity check** between the user and their friends. A proximity event occurs, when one user is inside the defined proximity radius of a friend. In this case the BuddyFinder service will inform the users by sending a proximity event.

At the client side, the session service is the central access point for XMPP connections of the client and maintains the communication with the MobilisBuddy service. The other services are the counterpart of the server-side services.

In the following, we show how the proximity updates are processed and how social networks are integrated into MobilisBuddy.

## Proximity Updates

The collaboration between clients is achieved using the services already shown. Those services communicate in a structured way. In particular, the MobilisBuddy system makes use of special XMPP data packets (Info/Query stanzas) to realize a request-response-mechanism. We present the realization of this proximity-based BuddyFinder service in the following example depicted in Figure 5.

It shows how Client1 changes his location and automatically sends a location update (LocationIQ SET) to the server. Using the Context Service and

*Figure 4. Structure of MobilisBuddy*

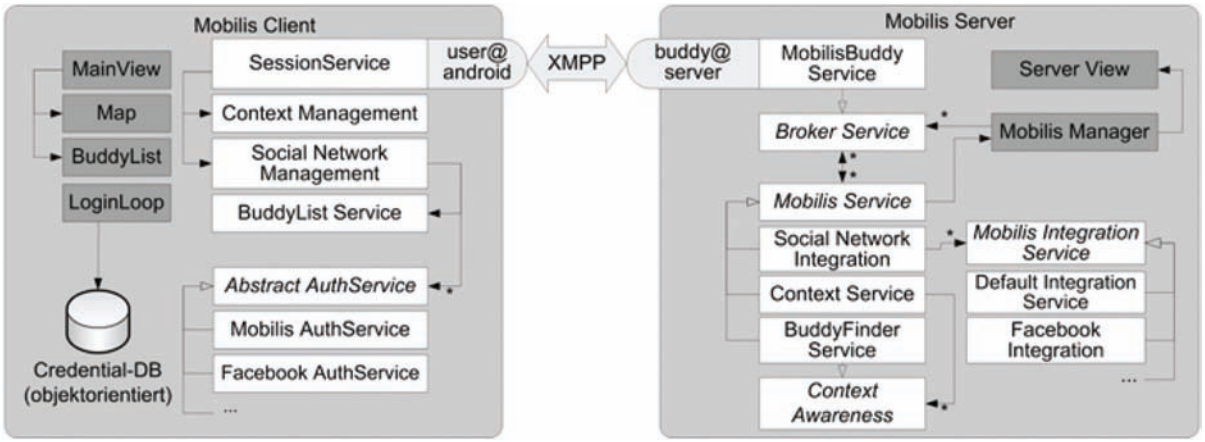
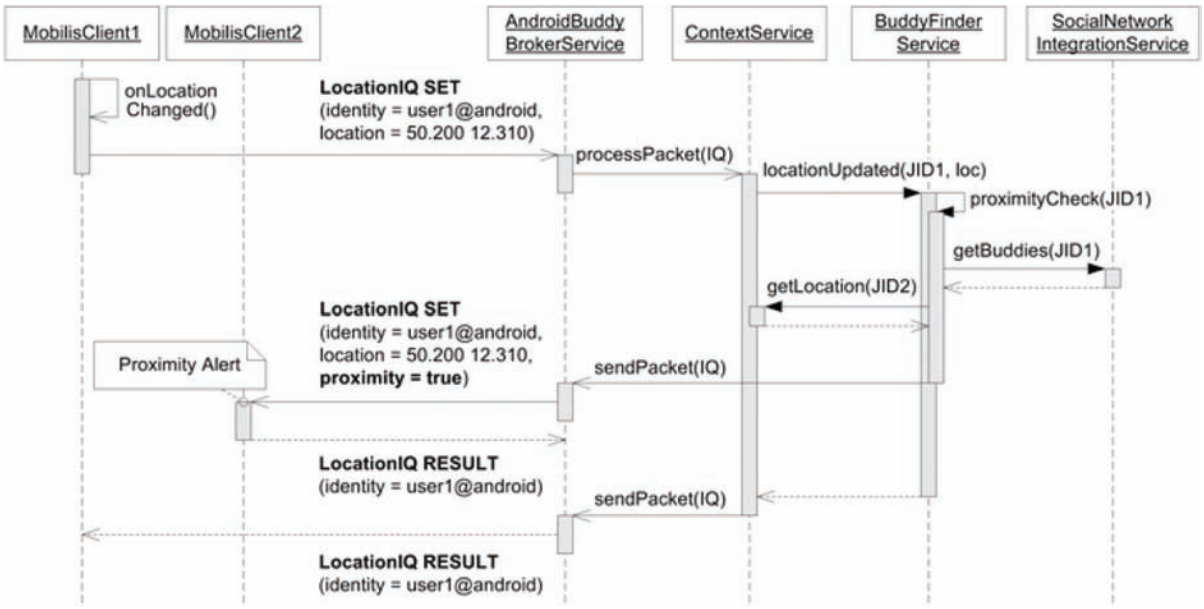


*Figure 5. Location updates and proximity check*




the Social Network Integration Service the BuddyFinder service then determines that Client1 is inside the radius of Client2. It thus sends a location update further to Client2 which will then show a proximity event on its screen. All issued Location Updates are confirmed by a LocationIQ RESULT.

## Integration of Social Networks

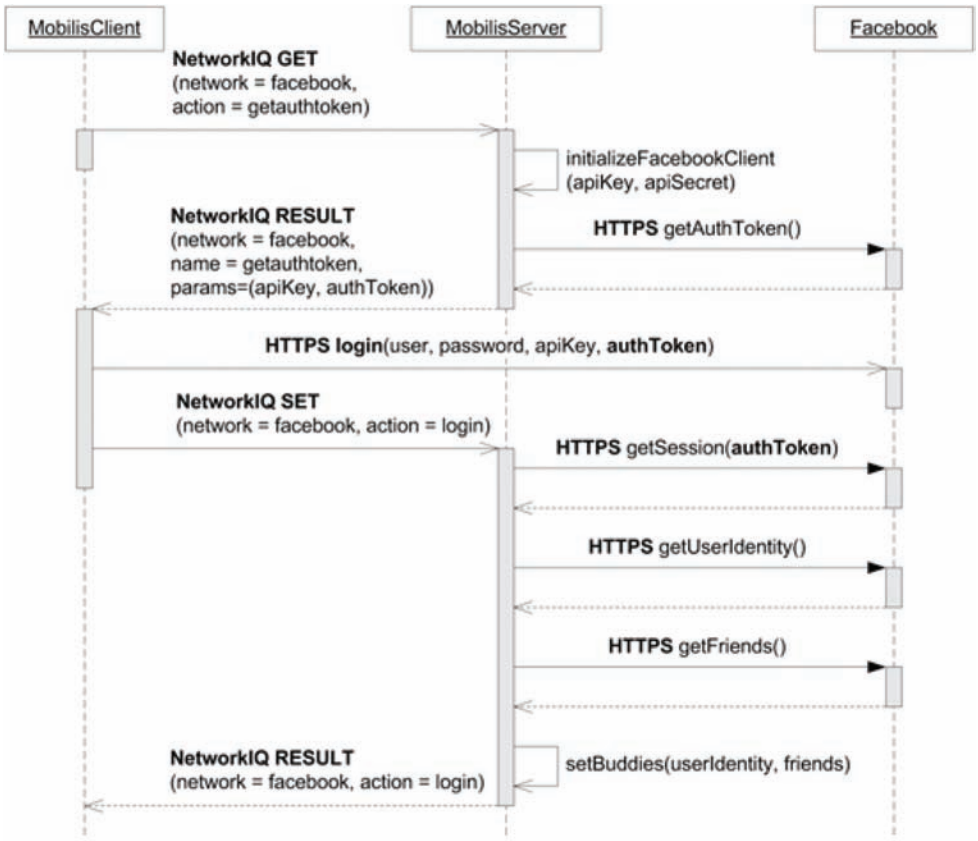
The system allows automatic login (LoginLoop) into various social networks using credentials stored inside a DB4O database on the mobile phone. This database is object oriented and optimized for lowest possible memory usage. It allows easy adding of new data structures.

The integration of information from one social network is realized by a Mobilis Integration Service on the server side. It manages a directory of user IDs (related to the respective social network) with their mapping to corresponding Mobilis Jabber-IDs. It also maintains a list of friends for each user. Hence, for every social network which should be integrated, a specific Mobilis Integration Service has to be developed. The Social Network Integration Service aggregates all Mobilis Integration Services. For one given user, it may request the friend lists from all connected social networks with each friend resolved to the respective Jabber-ID. It then returns an aggregated list to the user.

We want to outline the integration of Facebook as a social network in detail. It is realized by the Facebook Integration Service, which has access to the Facebook friend list of the user. This access is achieved by a desktop application registered at the Facebook platform. To allow maximum security, all user data needed to log in at Facebook is transmitted directly from the Mobilis client to Facebook and not over the Mobilis server. This concept of negotiating a Facebook session with Mobilis client, server and Facebook is depicted in Figure 6.

The client issues a connection request to the Mobilis server which requests an auth token from Facebook. Every request to Facebook will need the Facebook API key and API secret stored at the Mobilis server. The private API secret will not be communicated to the Mobilis client

*Figure 6. Integration of Facebook contacts*




whereas API key and auth token are transmitted. The client then logs in using the Auth Token, API Key and the private Facebook credentials and finally confirms a successful registration to the Mobilis server.

For efficiency reasons the further communication with Facebook is done by the Mobilis server. The server requests a session key from Facebook providing the auth token and API secret. It than has limited read access to the user account. The user which is now logged in at Facebook has a Facebook-specific ID, which can only be read by the Mobilis server using the API secret and the session key obtained during login. That way, the server knows the Facebook-ID and the Jabber-ID for every registered user and can perform a mapping between both. It then requests the user‘s friend list from Facebook. The friend list – also consisting of Facebook-IDs – is mapped to Jabber-IDs; only friends, which are also currently registered to the Mobilis system, are added to the friend list of the user.

Using this scheme it is possible to integrate contacts from multiple social networks and thus link the two worlds of social networks and mobile real-time collaboration.

## IMPLEMENTATION AND EVALUATION

We implemented the architecture described in the last Section and realized the applications MobilisGuide, MobilisBuddy and MobilisTrader within this architecture. In order to implement Android applications it is convenient to use Google’s An-

droid Eclipse plug-in. We will highlight some of the findings in the following.

## XMPP Interface Implementation

A Java-based Mobilis server has been developed. Both, Mobilis server and Mobilis client, communicate via XMPP which is powered by an additional XMPP server.

Openfire has been chosen as XMPP server. The XMPP connection on client and server side is realized through the Java-based Smack libraries. Integration of these libraries into the Android environment proved to be uncomplicated. Only small changes were necessary in Smack: Mapping XMPP messages to IQ objects was normally achieved by code introspection. This way led to errors on the Android device which is why explicit IQ providers had to be implemented. An additional library called su-smack (Su-Smack, 2009) has been chosen as an implementation of XEP-0060 (publish-subscribe) [Xep0060] but was perceived being immature. Overall, for the implementation of communication via XMPP standard libraries could be used, which are easily adaptable to Android.

## Facebook Interface Implementation

On the Mobilis server the Facebook Java API has been integrated to realize the communication between the Mobilis server and Facebook. Facebook published the API for external access to the social network, although it will not be developed any further by the company. Therefore, it became open source and a community is now responsible for maintenance. Only small changes in the authentication process had to be applied to use the Facebook API on the server.

On client side we did not choose to integrate the full Facebook API to minimize load on the client and traffic in the mobile network. Only the transmission of login credentials to the Facebook server gets initiated by the client for security purposes. This is realized by the so-called FacebookMockLoginBrowser logging into Facebook through a simulated HTTPS browser.

## Validation

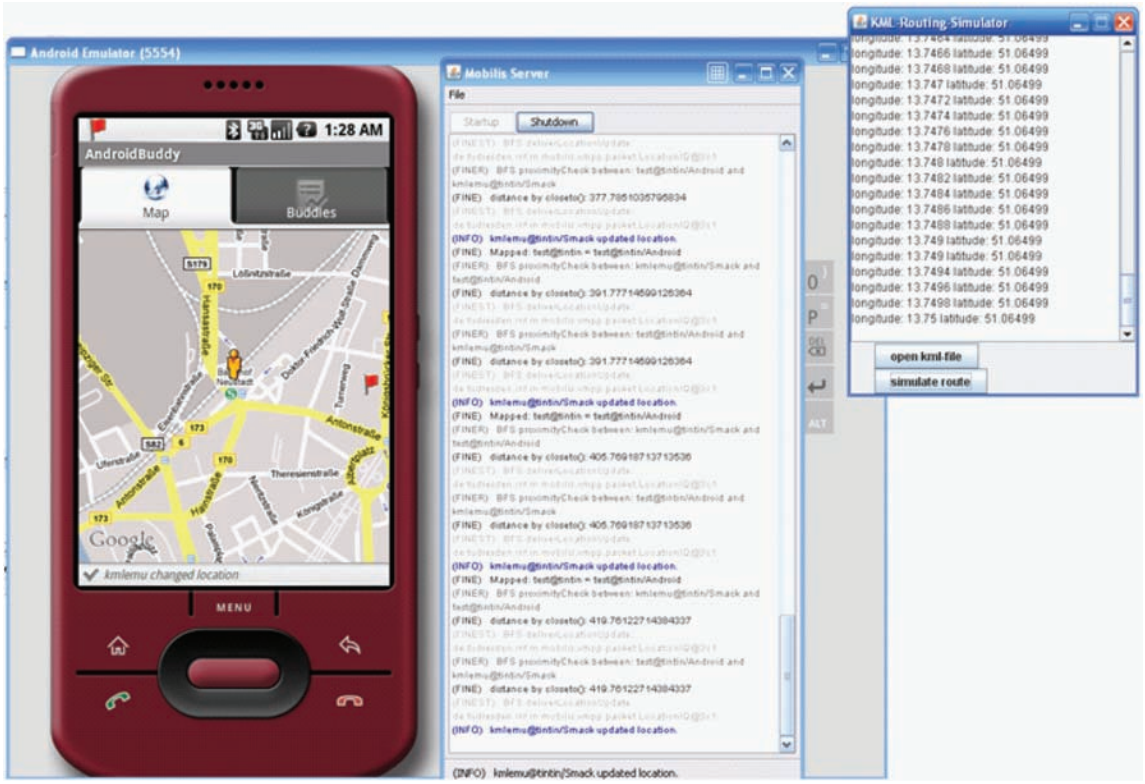
A functional test was accomplished at the MobilisBuddy application prototype. Thereby, different XMPP IQs were sent to the Mobilis system to check the system behavior against its specification. This black box test was performed in three steps.

At first, server and client interfaces were tested separately. For this purpose, we manually sent XMPP IQs to the JID of the MobilisBuddy BrokerService and the client instance respectively, via an XML console provided by the Miranda Instant Messenger. Test cases included for example logging into social networks and updating user location data.

The next step incorporated a typical system test, where multiple instances of the client and one instance for each server, namely the XMPP and Mobilis server, were started on different computers in the network. Changes of the client location were simulated by the Android emulator control integrated in Eclipse. Reactions of the server and forwarding of proximity events were carried out correctly. In case of many logged in clients the number of performed matching operations increased dramatically. This issue could be addressed in future by employing XEP-0060 (publish-subscribe).

In the end, an automated Keyhole Markup Language (KML) routing simulator was designed and implemented, which reads a KML file describing a path along geographical coordinates. This Java-based simulator connects to the Mobilis server like any other client and automatically sends location updates according to the KML path. In summary, this way enabled a fast and reproducible validation of the interfaces.

*Figure 7. Screenshot evaluation scenario*




## DISCUSSION

### Applicability of XMPP

Generally, all of our use cases of mobile collaboration were more or less straightforwardly mapped to functionalities provided by the pure XMPP protocol and its extensions. Even further application areas or requirements could be adopted because of the fundamental extensibility of XMPP, which allows the usage of both request/reply and publish/subscribe paradigms.

Regarding the currently available and standardized approach of establishing an XMPP session with a server, the first noteworthy property is the dependence on a persisting TCP connection, which is used for the basic streams wherein the particular stanzas are sent. First, open network sockets are very energy consumptive anyway because of the constant wireless connection between the mobile device and the base station. Second, if a network outage occurs, the stream exchange is immediately terminated, causing the server session to be closed.

When connecting again, the login procedure has to be repeated. On every session establishment, a verbose stream initiation process is started that includes Transport Layer Security (TLS) and Simple Authentication and Security Layer (SASL) negotiations demanding an amount of handshakes. Also resource binding and roster management demand stanzas to be received and sent out.

Especially because of the initial support of XMPP in the first versions of the Android stack, the community lately became aware of the drawbacks of the protocol used in a mobile environment. Hence this section presents the main outcomes of the discussion for the sake of completeness. As most of the proposed extensions were still in an

experimental or draft status at time of implementation and therefore subject to change, no reliable libraries were available.

The first extension reviewed here is XEP-0124 Bidirectional-streams Over Synchronous HTTP (BOSH) and introduces the idea of emulating a bidirectional TCP stream by using a synchronous HTTP connection without the need of polling. Thereby, sessions may be continued after short disconnections.

The most promising and efficient enhancement, particularly in respect of network outages, is XEP-0198 Stream management, which enables the server to distinguish between connection errors and voluntary suspends. The specification improves network reliability by the possibility of sending packet acknowledgements. Additionally, by standardizing the process of stream resumption, short network outages may be bridged without the need of a long and verbose reconnection phase.

This small overview shows that the XMPP community is aware of the mobility problems and is already working on solutions. As a future work, we plan to take part in this effort by providing measurements of the behavior of our applications in the case of network outages and poor Internet connections.

### Applicability of Android

The prototype application's user interface is mainly map based. By that, it reveals those parts of the presented application framework, which are mostly dependent on the propositions of the Android stack; namely all location-based views, services and sensors. Concerning this matter, the built-in libraries facilitated the development in many respects; for instance, the Location Manager was used for current location data from the GPS sensor and the Location class for simple distance measuring. Additionally, the Map View provided an easy customizable approach for overlaying icons. Indeed, the convenient location based services of the Android software stack are a significant advantage for employing the operation system compared to other mobile platforms available currently.

The other main concept introduced by Android and employed by the interaction between the Application Framework Layer and the User Interface is the Intent mechanisms. Intents are used for enabling multiple parts of the UI to react upon broadcasts without prior registration. Beyond, the broadcast approach allows other applications outside the scope of the framework to listen to these intents, provided that they deal with respective permissions. Further, common tasks like enabling the user to call a user out of the application can be easily allowed by using Intents.

## CONCLUSION AND FUTURE WORK

In this chapter we described our experiences with the implementation of mobile collaborative applications based on the open protocol standard XMPP. We introduced a general reference architecture introducing a set of Basic and Advanced (Mobilis layer) services and described how to use these services for implementing collaborative features in applications on mobile devices. Especially, we discussed the implementation of features like interest match, social network integration, proximity support, and group management as important aspects for mobile collaborative applications. As an efficient platform for the implementation of the concepts Android has been introduced and proven as the best choice among several platforms for application development for mobile infrastructures.

The platform described in this chapter provides a lot of research opportunities to be tackled in future work. The area of XMPP in mobile environments has to be investigated in more detail. It would especially be helpful to run large-scale simulations or real-world experiments to test XMPP behavior in the event of network connection loss, low data rate, or high round-trip times. Our platform provides the possibility to run real

collaborative applications on top of XMPP and to develop solutions for these problems.

Furthermore, the service-based approach should be elaborated more in future work. We envision an open XMPP-based service environment, where developers can easily publish their services to be reused by other developers. The service environment we presented in this paper is a good first step in this direction. But it would be helpful to engage more developers of real-time collaboration applications for mobile devices in the process. We plan to make our project open source to foster this type of collaboration.

It was shown in the chapter how coupling to existing social networks can be realized and that it provides an added-value for collaborative applications. If we think one step further in this direction, the platform should provide coupling to any type of Web 2.0 application such as social networks, blogs, forums, media sharing portals or the upcoming Google Wave to enable pervasive collaboration regardless of the type of device currently in use. This coupling should be done in a flexible but secure way to enhance the user base of collaborative mobile applications but to keep full control of what personal information to share with whom. This will surely be one of the biggest challenges of future pervasive collaboration.

## ACKNOWLEDGMENT

The Mobilis framework was created by the cooperating institutions TU Dresden, PUC Rio de Janeiro, and UFMG Belo Horizonte in the project Mobilis partially funded by the German BMBF and the Brazilian CNPq. We want to thank all the student developers of the prototypes as well as all other project partners for the fruitful discussions that led to the Mobilis architecture and the work presented in this chapter.

## REFERENCES

Android Developers. (2009, November 25). *Intents and intent filters*. Retrieved from http://developer.android.com/guide/topics/intents/intents-filters.html

de Poot, H., Mulder, I., & Kijl, B. (2005). How do knowledge workers cope with their everyday job? The *Electronic Journal of Virtual Organizations and Networks, 9*, 70-88. Retrieved from http://www.ejov.org/apps/pub.asp?Q=1643

Franz, E., Groba, C., Springer, T., & Bergmann, M. (2008). A comprehensive approach for context-dependent privacy management. *Proceedings of the Third International Conference on Availability, Reliability and Security (ARES 2008),* Barcelona. doi:10.1109/ARES.2008.184

Google, Inc. (2009, November 25). *Google app engine – Google code*. Retrieved from http://code.google.com/intl/en/appengine/

Groba, C., Groß, S., & Springer, T. (2007). Context-dependent access control for contextual information. *Proceedings of the The Second International Conference on Availability, Reliability and Security*, (ARES 2007), (pp. 155-161). doi: 10.1109/ARES.2007.61

Hildebrand, J., Millard, P., Eatmon, R., & Saint-Andre, P. (2008). *XEP-0030: Service discovery*. Retrieved November 25, 2009, from http://xmpp.org/extensions/xep-0030.html

Hutton, J. (2001). *Unified communications - A potential cure for communications overload*, (p. 50). CMA Management, October 2001. Retrieved November 25, 2009, from http://www.highbeam.com/doc/1G1-78974004.html

Millard, P., Saint-Andre, P., & Meijer, R. (2008). *XEP-0060: Publish-subscribe*. Retrieved November 25, 2009, from http://xmpp.org/extensions/xep-0060.html

Open Handset Alliance. (2009, November 25). *Open handset alliance website*. Retrieved from http://www.openhandsetalliance.com/

Pallickara, S., & Fox, G. (2003). NaradaBrokering: A distributed middleware framework and architecture for enabling durable peer-to-peer grids. In M. Endler (Ed.), *Proceedings of the ACM/IFIP/USENIX 2003 international Conference on Middleware* (Rio de Janeiro, Brazil, June 16 - 20, 2003) Middleware Conference, (pp. 41-61). New York, NY: Springer-Verlag.

Saint-Andre, P. (2008). *XEP-0045: Multi-user chat.* Retrieved November 25, 2009, from http://xmpp.org/extensions/xep-0045.html

Sørensen, C., Mathiassen, L., & Kakihara, M. (2002). Mobile services: Functional diversity and overload. *Proceedings of New Perspectives on 21st-Century Communications*, Budapest, Hungary, 2002, (pp. 1-12).

Springer, T., Schuster, D., Braun, I., Janeiro, J., Endler, M., & Loureiro, A. A. (2008). A flexible architecture for mobile collaboration services. In *Proceedings of the ACM/IFIP/USENIX Middleware '08 Conference Companion* (Leuven, Belgium, December 01 - 05, 2008), Companion '08, (pp. 118-120). New York, NY: ACM. doi: 10.1145/1462735.1462770

Su-Smack. (2009, November 25). *Su-smack website*. Retrieved from http://static.devel.it.su.se/su-smack/

XMPP. (2009, November 25). *Website of the XMPP Standards Foundation.* Retrieved from http://xmpp.org/

## KEY TERMS AND DEFINITIONS

**Android:** Mobile operating system running on a Linux kernel. Android is maintained by the Open Handset Alliance led by Google.

**Mobile Collaborative Applications:** Collaborative applications are pieces of software to help people engaged in common task to achieve their goals. These applications are also called groupware while we prefer the term collaborative application or collaboration applications. The word "mobile" refers to such applications used on mobile devices such as mobile phones or PDAs. While notebooks may also be considered mobile devices, we focus on resource-restricted devices like mobile phones in this chapter.

**Proximity-Based Services:** Subclass of location-based services, i.e., services accessible with mobile devices through a mobile network making use of the geographical position of the device, especially the geographical proximity of two or more devices.

**Service Environment:** Logical architecture comprising client and server components (services) including standard interfaces and protocols as well as mechanisms for registering and finding services and their properties.

**Social Network:** Web 2.0 application building online communities of people who share interests and/or activities.

**XEP:** XMPP Enhancement Proposals are specifications for extensions of XMPP written in the same format like the XMPP core standards but not standardized within the IETF. The XEPs are reviewed by the XMPP Standards Foundation (XSF) and published on its website.

**XMPP:** The eXtensible Messaging and Presence Protocol is a family of XML-based network protocols for open real-time communication and collaboration standardized by the IETF and further developed by the XMPP Standards Foundation (XSF).