A Framework for the Development of Mobile Social Software on Android

Robert Lübke, Daniel Schuster, and Alexander Schill

Computer Networks Group, Technische Universität Dresden, Dresden, Germany,
{robert.luebke,daniel.schuster,alexander.schill}@tu-dresden.de

Abstract. Nowadays, social aspects can be found in more and more mobile applications. Analyzing the domain of mobile social software and especially the offered features shows much similarity in the different apps. A framework that supports often used functionalities can facilitate and speed up the development of new innovative mobile applications. This paper presents the Mobilis framework, which aims to extensively support the developers of mobile social software. It provides often used features and offers the functionality in a service environment, that can easily be integrated into new mobile applications.

Key words: social software; mobile computing; framework; social network; group formation; collaboration; XMPP; Android

1 Introduction

In the last years social networks have gained a huge popularity. Nowadays they connect millions of internet users with each other. Facebook Places[1], Foursquare[2] and Gowalla[3] are well-known examples of apps that combine social networking and pervasive computing and thereby form the trend of *pervasive social computing*. While many social platforms are accessible via apps for mobile devices, they do not unfold their full potential until the connected users can interact and collaborate with each other in a way that is adapted to the current situation and technical infrastructure. With collaborative features users interact with each other in a target-oriented way to reach a common goal and they coordinate each other with their mobile devices. Such collaborative social apps can be applied in the private and the business sector. However, the development of those collaborative services is quite expensive. There are various other features of mobile social software that should also not be designed and implemented from scratch in every new app.

This paper presents the Mobilis framework, that aims to overcome this issue by extensively supporting the developers of mobile social software. It consists of client and server components and basic XML-based communication mechanisms

[1] http://www.facebook.com/places
[2] http://www.foursquare.com
[3] http://www.gowalla.com

between those components. It provides features, that are often used in mobile social applications. The functionality is offered as parts of a service environment, which can be easily integrated into new mobile applications. These services do for instance cover (location-based) group formation, user context management, integration of social networks, collaborative editing of documents and media sharing. Furthermore it provides an infrastructure for easy communication within social networks. To our knowledge until now there is no other system supporting the developers of mobile social software with these needed special functions, neither in research nor in commerce.

After giving an overview about the domain of mobile social software and discussing related work, we define the requirements of the framework for mobile social software. We further discuss possible technologies to use in this service environment. The Mobilis platform itself is presented in the following sections. At the end the results are evaluated by showing apps that were developed using the Mobilis framework.

2 Background and related work

Mobile social software is a class of applications for mobile devices with the main goal to allow and support the social interaction in a community of mobile users. Mobile social software enables communication, coordination and collaboration within social networks.

There is a large number of applications in this domain. Hence there are multiple attempts to create classifications and taxonomies, in which the mobile social apps can be arranged.

According to [1] there are three social network models: *intimate*, *crowd* and *hybrid*. In intimate social networks the users interact with people they have close personal connections to, for example friends and family. In crowd networks on the other hand the interaction is among users who are unknown to each other. A good example for these networks is the auction platform eBay[4]. Hybrid networks combine aspects of both social network models. In systems such as Flickr[5] the user can benefit from personal connections, but one can also use content generated by the user crowd.

In [2] mobile social software is classified depending on the social context. The authors consider four dimensions of social context: *spatial*, *temporal*, *inference* and *people* (STIP). The spatial dimension defines how relevant the geographical distance of the users is. In the temporal dimension the authors differ between short-, mid- and long-term social interactions. The inference dimension describes how the social context of the user is inferred. Finally the people dimension represents the kind of the counterpart the users interacts with. These can be individuals, groups or anonymous communities. All systems are analyzed according to these dimensions and given a characteristic STIP vector.

[4] http://www.ebay.com
[5] http://www.flickr.com

One can also classify the applications according to their main functionality. Many apps allow the sending of media or simple text messages like Twitter[6]. Other systems notify the user if a friend or a special event is in the vicinity. Examples for these proximity-based services are VENETA [3], WhozThat [4] and PeopleTones [5]. Urbiflock [6] and Socialaware [7] are systems that exploit location context to establish user groups. Geotagging apps like Google Buzz[7] form another class. One can also find many dating apps, for example MeetMoi[8]. There are also Mobile Social Games and apps with gaming features like Foursquare and Gowalla.

In summary, many mobile social applications often show similar or even identical functionalities like communication, media sharing, grouping and collaborating. The notion of location also plays an important role in most applications. These features should not be designed and implemented from scratch in every new mobile app. Hence there is the need for a framework supporting these often used functions.

Different existing frameworks and middleware layers like SAMOA [8], MobiSoc [9] and MobiClique [10] facilitate the development of new applications. These systems have the main goal to divide the user community into multiple social networks. For this purpose user and location profiles are applied and matched with each other to detect interesting persons and places for the user. All of the mentioned frameworks and middleware layers cover the details of social networking while app developers can concentrate on their application logic. But in essence they do not support the mentioned typical functions that are used within a social network. Therefore we want to discuss the requirements, the conceptual design, implementation details and evaluation aspects of the mobile social software framework Mobilis, that provides exactly this often used functionality.

3 Requirements

The previous section showed that mobile social software often has the same or similar features. Therefore a framework providing these features would be useful. In this section the required features are analyzed.

The framework should support simple communication mechanisms for all involved entities (*FR-1*). Especially the users of the applications must be able to exchange text messages. Furthermore all communication partners should be able to exchange and fetch presence information (*FR-1.1*). Every user can set his own presence status (available, busy, do not disturb, etc.) and propagate it to a presence server. Additionally, groups of users must be able to communicate in multi-user chat rooms (*FR-1.2*).

Context information of the user (current location, friends, used device, etc.) play an important role in the area of mobile social software, for example to make

[6] http://www.twitter.com
[7] http://www.google.com/buzz
[8] http://www.meetmoi.com

applications adaptable. Therefore the framework should determine and manage the different kinds of context information. The collected data can also be shared with authorized entities similar to a notification service in a publish/subscribe scenario (*FR-2*).

It is a common feature of mobile social software to use existing social networks like Facebook or Foursquare. The framework should support developers with the integration into own applications (*FR-3*).

The framework should offer file sharing functionality (*FR-4*), because this is often used within social networks. Mainly photos and videos are shared with friends. Therefore the framework should also support the more special sharing of media. Those media files can be tagged with meta information (recording time and location, author) that help searching in large collections.

Another main component in social networks is the formation of groups with users of similar interests. The claimed group management (*FR-5*) includes creating and deleting of groups but also the management of members and joinings. There is a special kind of grouping in the context of mobile social software. It is called location-based group formation and it allows the users to see and join special groups only at certain places.

The framework should support a matching mechanism (*FR-6*), that can be used based on proximity, interests, preferences and activities. Proximity detection informs the user about possible actions in his near surrounding. An application that matches locations of entities could for instance raise an alarm if one of the user's friends is near or if a near restaurant has a special offer. Another example is a trading function based on matching of requests and offers. It could for instance be used in a shopping application, that allows to offer and search for products in a social network.

Real-time collaboration within social networks gains more and more importance. One reasonable collaborative function is the shared editing of documents or other shared state. The framework should support simultaneous editing of XML-based documents by multiple users (*FR-7*). Thus, developers of mobile social apps can utilize this function for easily managing any structured multi-user objects like game state, a shared playlist or shared task list. The mentioned features were already implemented in some of the systems we surveyed (yet without using a shared editing service).

All of the above mentioned functions shall be implemented as reusable services. So the service paradigm should be realized (*FR-8*). As in common service-oriented architectures the functionality is encapsulated and should be accessible via well-defined interfaces (*FR-8.1*). The usage of these services has to be independent from the underlying implementation (*FR-8.2*). Furthermore, such a service environment needs to have mechanisms for service discovery (*FR-8.3*), the different entities can use to find the functionality they are looking for. Additionally the framework should support different versions of the various services (*FR-8.4*).

Table 1. Requirements

Identifier	Description
FR-1	Communication
FR-1.1	Management of presence information
FR-1.2	Multi-user chat
FR-2	User context management
FR-3	Integration of existing social networks
FR-4	File and media sharing
FR-5	Group management (based on location)
FR-6	Matching mechanism
FR-7	Shared editing
FR-8	Realize service paradigm
FR-8.1	Encapsulate functionality; well-defined interfaces
FR-8.2	Transparency to underlying implementation
FR-8.3	Service discovery
FR-8.4	Support of different service versions

Table 1 once more summarizes all the mentioned functional requirements we derived from our survey of current mobile social apps. It represents a good core of services covering most of the commonly used functionality in the mobile social apps we surveyed. As such a list can never be complete in terms of covering every reusable functionality, an open, extendable approach is needed for the architecture of the platform. In the following, we first survey appropriate technologies for this purpose before defining our architecture.

4 Technologies

This section presents details about the technologies used to realize the service environment of the framework. Many existing solutions use proprietary protocols. While this may be an efficient solution, they mostly depend on a special mobile platform. Thus, interoperability is not given and the degree of reusability is low. Therefore a fundamental requirement is to use standardized solutions. The technology should also have an active community adapting it to recent developments.

One possible solution is to implement the functionality in web services and to let client and server communicate via *SOAP*. However, SOAP can not meet the main requirement of easy communication among all clients (see *FR-1*).

The *Session Initiation Protocol* (SIP) meets the stated demands. Especially the mechanisms of the registrar servers and the location service make SIP a convenient protocol for mobile environments. SIP is appropriate for the given use case, but unfortunately most providers block all SIP messages in their networks to avoid VoIP traffic. It would be counterproductive to apply a protocol, that can not be used in all mobile communications networks, as a basis for innovative

mobile apps. Furthermore SIP has a big overhead due to the protocol headers in every message.

The eXtensible Messaging and Presence Protocol (XMPP) [11] is an XML-based protocol collection for the exchange of messages and presence information, that also meets the requirements of the service environment of the framework. Originally the project was called Jabber which was developed from 1998 to 2000. In 2002 Jabber was standardized by the Internet Engineering Task Force (IETF) under the name XMPP.

One of the strengths of XMPP is security, because the core standard includes multiple mechanisms for encryption and authentication. For example one can use the *Simple Authentication Security Layer* (SASL) to authenticate and *Transport Layer Security* (TLS) for encryption.

Each XMPP entity has its own XMPP address, that can be used to contact this entity. It is also called XMPP-ID, Jabber ID oder JID. A JID is composed similarly to an email address, because it has the form `username@XMPP-server`. Of course the user has to possess a registered account at the specified XMPP server. XMPP also supports multiple simultaneous connections over one account. A special resource identifier is used to differentiate between the connections. It is appended to the *bare JID*. The resulting form `username@XMPP-server/resource` is also called *full JID*.

At first an XMPP client has to log in at his XMPP server. After establishing a TCP connection with two XML streams – one for each communication direction – information can be exchanged via well-defined XML stanzas. There are three different kinds of XML stanzas in the XMPP standard.

The *message stanza* can be compared to a push mechanism. Mostly it is used for instant messaging among the users as it can contain text messages. But it can also be used for notifications about special events.

The *presence stanza* contains presence information about XMPP users. This is realized with a publish/subscribe mechanism, i.e., multiple entities can receive status updates about the entity they are subscribed to. These status updates are sent to all subscribers like in a broadcast. A presence stanza can also contain priorities besides the textual description of the status (available, do-not-disturb, away, etc.).

The *info/query stanza* can be used to realize simple request/response services within the XMPP architecture. Like in a remote procedure call, one entity can send a request to another entity, that processes this request and then sends back a result or an error message. Therefore an IQ stanza has one of the following four types: `GET` to request information, `SET` to determine values, `RESULT` to return the outcome of a request or `ERROR` to report on maloperation. The XML structure of an IQ with all necessary elements and attributes is shown in Figure 1. It has an `iq`-element as its root. Beside the `type` every IQ has an identifier (`id`), a sender (`from`) and a recipient (`to`). Below the root element one can find the child element with the namespace (`xmlns`). The part consisting of the child element and all its children is also called payload of the IQ.

```
<iq id="abc123" type="get"
    from="client@xmpp/MyCar"
    to="NavigationSystem@xmpp/RouteCalculation">
    <query xmlns="example.com#services/RouteCalculation">
        <target-location>
            <longitude>13.766044</longitude>
            <latitude>51.036649</latitude>
            <altitude>45</altitude>
        </target-location>
        <target-name>
            Potsdamer Platz 1, Berlin, Germany
        </target-name>
    </query>
</iq>
```

Fig. 1. Example of an Info/Query (IQ) stanza.

Another major strength of XMPP is its extensibility. All three stanza types can be extended regarding various aspects. One can define and use own protocol extensions, but there are also over 200 existing XMPP Extension Protocols (XEP) that were developed and later standardized by the XMPP community. Some of the most useful ones for mobile social apps are multi-user chat, service discovery, and publish/subscribe, which will shortly be introduced in the following.

XEP-0045 [12] defines the XMPP extension protocol for multi-user chat. Like every other XMPP entity, a multi-user chat room has its own XMPP address. The participants have to register with the chat room, to take part in the conversation. Group messages are sent directly to the chat room JID and then forwarded to all participants. The Service Discovery extension XEP-0030 [13] covers finding information about special entities in the XMPP network, for example supported protocols, features, extensions and offered services. This XEP is very important in service environments like the one that should be developed for the mobile social software framework. XEP-0060 [14] defines the functionality of a generic publish/subscribe service, that can be employed in various use cases. One user publishes a certain piece of information and a notification about that event is sent to all authorized subscribers. XEP-0060 also defines how the subscription of such information updates and the authorization is done.

There are still other XEPs that could be interesting in mobile social software, especially those extensions, that define a certain payload format to exchange information about the user, for example his location [15], his mood [16] or the music he is currently listening to [17]. It was already pointed out before, that file sharing, which is covered by XEP-0096 [18], is an important feature in social networks, too.

In summary, XMPP suits well as a technology for the service environment of the mobile social software framework. Especially the IQ mechanism enabling simple request/response services will be used in the framework. XMPP also supports easy communication between the entities (see *FR-1*). XMPP is based on XML and therefore platform independent. Many of the mentioned standard

extensions are applicable in mobile social software. The XMPP standard includes security aspects that have even more importance in mobile environments.

The disadvantage of high overhead is due to platform independent coding in XML format. So it is not a special disadvantage of XMPP but one of all platform independent service environments. Extensions for XML stream compression [19] help to overcome this issue.

5 The Mobilis platform

This section gives a detailed overview about the Mobilis platform - a framework for mobile social software. After describing the general architecture, some of the main components are discussed in detail.

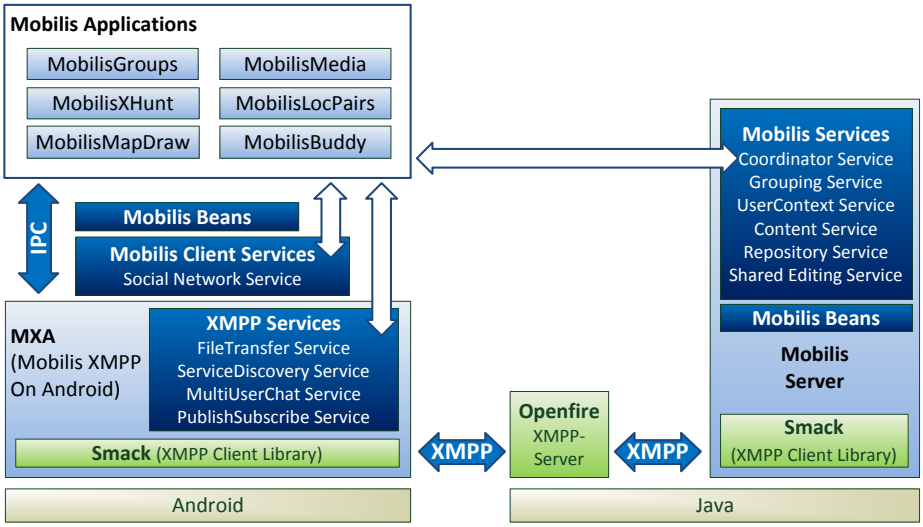

Fig. 2. General architecture of the Mobilis platform.

5.1 General architecture

The general architecture of the Mobilis platform is shown in Figure 2. The MobilisServer offers various services, the so-called Mobilis Services, to the client applications. These services have their own XMPP addresses and can be called via XMPP messages. The communication is based on the XMPP client library Smack[9]. It facilitates sending and receiving of XMPP packets within Java applications. We use Openfire[10] as XMPP server, but every other XMPP server

[9] http://www.igniterealtime.org/projects/smack/
[10] http://www.igniterealtime.org/projects/openfire/

could also be employed as we do not extend the XMPP server implementation itself.

To assure the communication on the client side several adjustments had to be made to Smack to use it within Android applications. Therefore the functionality of Smack is encapsulated in the application MXA (Mobilis XMPP on Android). This application provides an AIDL (Android interface definition language) interface to the services of MXA. In this way MXA can be used by the different Mobilis applications to communicate via XMPP with the MobilisServer.

Beside the basic communication, MXA provides additional XMPP Services. These services realize certain XMPP extensions. One service enables file transfers within the XMPP architecture, another provides administrative functionality for multi-user chats. It is also possible to use XMPP service discovery and a publish/subscribe mechanism.

The component Mobilis Beans is a collection of structures that help to create the custom XMPP messages, which are exchanged between the MobilisServer and the clients. Therefore the Mobilis Beans contain the custom protocols and define how to use the services of the MobilisServer from the Mobilis application's point of view.

Developers of Android applications can find special services in the Android Library Mobilis Client Services. These services are components that are often used in mobile collaborative and social software. One of these services is the Social Network Service, that facilitates the integration of existing external social networks into own applications.

5.2 Mobilis Client Services

Developers of Android applications can find special services in the Android Library *Mobilis Client Services*. These services are components that are often used in mobile collaborative and social software.

At the moment one can see that existing social networks more and more link with each other. Therefore it should be possible for new social applications to integrate existing networks. Furthermore new social apps suffer from small user numbers. A real networking among the users is not possible until a critical mass of participants is reached.

For these reasons the integration of existing external social networks into own applications is the main task of the Mobilis Client Services and especially of the Social Network Service. This service encapsulates the application programming interfaces (API) of different social networks and provides the developers with this functionality conveniently at one central place. The general architecture of the Mobilis Client Services is illustrated in Figure 3.

The Social Network Service is implemented as an Android Service, that provides its functionality via an AIDL interface. It manages multiple *Social Network Manager* components, which encapsulate the respective API functions.

At the moment the supported systems are *Foursquare*, *Gowalla* and *Qype*[11]. Foursquare and Gowalla are mobile social networking applications with gaming characteristics. Users tell the system where they currently reside by checking in at special locations. This is rewarded with points or bonus items. The most frequently visiting user becomes the virtual mayor of a place. Both APIs support vicinity search within the recorded locations, that can also be integrated into own applications. Qype is a recommendation system, in which the users can evaluate companies, locations, services etc. Adapters for other services like Facebook or Twitter can be added accordingly.

Applications have to authorize before they can use the APIs of the different social networks. Most systems use *OAuth*[12] for this purpose. With this protocol a user can authorize an application to access data on the social network system, without revealing the user credentials. The OAuth component of the Social Network Service manages this authorization process.

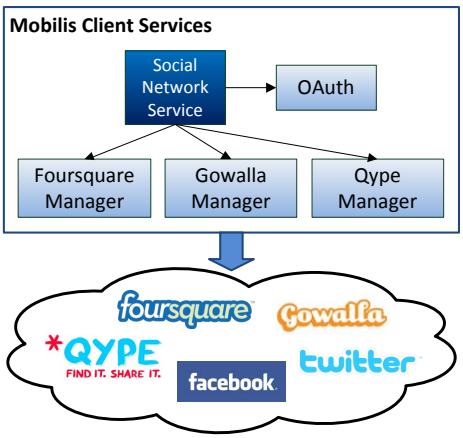

Fig. 3. Concept of the Mobilis Client Services.

5.3 XMPP Services

Several adjustments to the XMPP library Smack have to be made to use it within Android applications. Therefore the functionality of Smack is encapsulated in the application MXA (Mobilis XMPP on Android). This application provides an AIDL interface to its services that can be used by the different Mobilis applications to communicate via XMPP with the MobilisServer.

Beside the basic XMPP communication function MXA supports some special XMPP extensions, that are listed in Figure 4. One of these extensions is the multi-user chat, that is specified in [12]. With the help of the *MultiUserChat*

[11] http://www.qype.com
[12] http://oauth.net/

Service, one can use the basic functions like joining and leaving such a chat room, reacting on invitations, retrieving detailed information about a chat room and sending group messages. But it also supports various administrative functions like creating a new chat room, inviting users and kicking participants.

The *FileTransfer Service* enables file sharing within the XMPP network. It is based on XEP-0096 SI FileTransfer [18]. Users can send files and manage the incoming file tranfers.

The *ServiceDiscovery Service* realizes XEP-0030 that is specified in [13]. An application can find out more information about other XMPP entities with this service. That covers supported protocols, features and extensions of the respective XMPP entity. The service is used within the Mobilis platform to find all services that are currently running on a MobilisServer.

The *PublishSubscribe Service* realizes parts of XEP-0060. This extension specifies the basic functionality of a generic publish/subscribe mechanism. The PubSub Service supports subscription and cancellation of information updates.

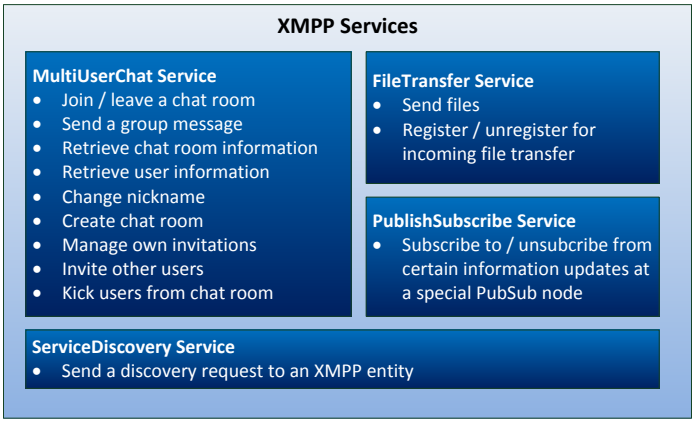

Fig. 4. Functionalities of the XMPP services.

5.4 Mobilis Services

The MobilisServer offers various services to the client applications. These so called Mobilis Services are the main component of the Mobilis service environment. Figure 5 illustrates the different layers the services can be divided into.

Client applications can get information about all active and supported services on a MobilisServer with the help of the *Coordinator Service*. This is done with an own XMPP protocol extension that is based on XEP-0030 (Service Discovery). It extends this XEP for instance with version numbers for the services. The Coordinator also allows to start new service instances. The client application only has to send the corresponding IQ stanza with the namespace of the required service and the Coordinator starts a new instance.

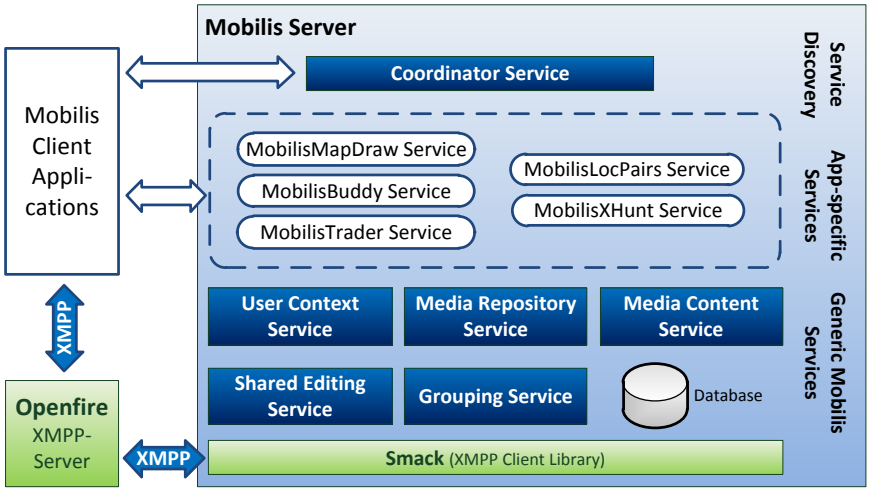

Fig. 5. Service layers of a MobilisServer.

The generic and reusable services are the basis of the MobilisServer. The most important services are presented in the following:

User Context Service The User Context Service manages all kind of context information of the users. Context covers all information that can be used to characterize the situation of a certain entity. Mostly these entities are people, but one can also think of places or other objects that can be relevant for the interaction of user and application [20]. The physical context for instance includes temperature, light and sound intensity of the user's surroundings. Examples of technical context are the characteristics of the network (bandwidth, latency) the user is connected to and also information about the mobile device he is using. Every user also has a personal context. This covers for example his address, phone number, calendar and his favorite payment method. Furthermore one can define the user's social context. This includes information about friends, family and groups of the user.

The User Context Service is designed to support all these different kinds of context information. It manages a so called context tree for every user. The nodes of this tree contain related pieces of context information about the user, for example his location, the music he is listening to or his mood. The nodes consist of simple key/value pairs. Additionally one can find the date of the last time the node was updated and the date it expires. After expiration of validity all data of the node is deleted. This assures a secure handling of possibly confidential context data.

The User Context Service is based on a publish/subscribe mechanism. The user decides which information to publish by sending it to the service. He also has to authorize all entities that should be able to access his data. All subscribers of a special node of the context tree are notified in real time if the data was updated.

Grouping Service The Grouping Service implements the functionality for the management of user groups, but it also includes location and time. This idea is called location-based group formation and discussed in detail in [21]. Groups are a very important feature in social software. Users with the same interests can find each other, discuss the group's topic, share photos, etc. But groups do mostly have spatial and temporal aspects like a special place and time of a meeting. Applying this idea in mobile social networks links the physical and the virtual world by creating incentives to be at a certain place at a certain time.

Repository and Content Service Media sharing is also an often used feature in social software. Mostly photos are exchanged among friends. The Repository Service and the Content Service realize this within the Mobilis platform. It furthermore specializes in sharing of photos with tagged geographical information. Users can for example search photos by location, time and author.

Collaborative Editing Service This service enables the shared and simultaneous editing of data by multiple users at different locations. This is also called Collaborative Editing or Shared Editing. The aim is to let all participants of a collaborative editing session see each other's changes in a document in real time. Possible conflicts during editing are detected and resolved. The Collaborative Editing Service supports any XML formatted document.

Application-specific services can be found in the layer above the generic services. These parts are not reusable because they do only serve special applications, but they can access and reuse functionality of the *Generic Mobilis Services*. They mediate between mobile apps and the generic services. Client applications can either directly contact and use the Generic Mobilis Services, or they can have an own App-specific Service that includes additional application logic. Examples for App-specific services are XHunt and LocPairs service. These are the server components of two location-based games that have been implemented with Mobilis (further details in section 6). The MobilisBuddy Service realizes proximity detection.

All services on a MobilisServer have their own XMPP address. That allows client applications to contact and use the services' functionality. Each service therefore has its own connection to the XMPP server. The discussed splitting of functionality into different services enables the deployment in distributed scenarios. Each service can run on its own machine without any further adjustments.

5.5 Mobilis Beans

It was already discussed in detail that the communication within the Mobilis platform uses XMPP and especially the IQ stanzas. Developers of new applications should not always implement the creation and parsing of IQ stanzas on their own. This is the task of the *Mobilis Beans* component, that contains various structures for the creation of XMPP stanzas, which are exchanged between client and server. Here one can find the particular parts of the protocols, that

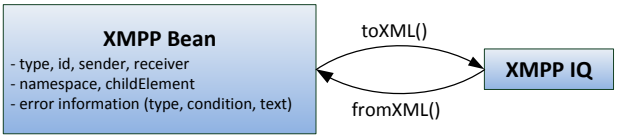

Fig. 6. Overview of important attributes and methods of Mobilis Beans.

define how to use the services of the framework. Developers of applications can use the Mobilis Beans to start requests to the services.

The main advantage is the reuse of the Mobilis Beans in all server and client components. Both parts of a Mobilis application reference the Mobilis Beans project and use parts of it for the communication among each other. This furthermore assures the same stanza syntax on both sides.

Each Bean represents a certain XMPP IQ stanza. It contains methods for the conversion into an XMPP IQ to send it, but there are also methods to parse incoming IQs into the correspondent Bean (see Figure 6). Furthermore it contains typical attributes of an IQ, for instance sender, receiver, namespace and child element. If it represents an IQ of type `Error`, it also carries detailed information about the occurred malfunction.

5.6 Web-based access

Mobilis does not only provide access to its services for native Android apps but also for web browsers. A web gateway to the Mobilis service environment enables access from PCs and Notebooks, but it also opens up other mobile platforms like iOS or Windows Phone. With new APIs introduced with HTML5 web applications can use more device functions. Furthermore new versions of JavaScript engines with increased performance allow designing impressive user interfaces for the web browser - even on mobile devices.

Using XMPP as communication protocol for the service environment of Mobilis leads to two major problems concerning the web-based access. First, the Hypertext Transfer Protocol (HTTP) is inadequate for bidirectional real-time communictation between client and server. Second, there are no APIs for native XMPP communication in the web browser. The first problem will be solved by the WebSocket API of HTML5 in the near future. But it is not appropriate to use this technology at the moment because of open security issues and the insufficient support by most of the web browsers.

Therefore we reuse existing established technologies to emulate the semantics of a long-living bidirectional connection. One of these technologies is *Bidirectional-streams Over Synchronous HTTP* (BOSH) that is specified in XEP-0124 [22]. It uses synchronous HTTP request/response pairs and no frequent polling to realize the bidirectional connection. The Connection Manager, that is part of the web server, administrates the long-living connection by establishing and releasing it.

A solution to the second problem is given in XEP-0206 [23] with the title *XMPP Over BOSH*. With this extension XMPP stanzas can be transfered as

child elements of the `<body>` element via BOSH. In the web browser a Javascript BOSH Client can extract and process the payload information. Combining this idea with the long polling paradigm of XEP-0124 enables bidirectional XMPP real time communication over HTTP.

With this mechanism the presented Mobilis Services (see section 5.4) can be used by client-side web applications that are based on JavaScript. A convenient level of abstraction is given, because the Mobilis JavaScript client library hides all the details of connection management and exchanging XMPP stanzas. It also has a plugin interface and can thereby be extended with further functionality.

6 Evaluation

The previous sections highlighted conceptional aspects and implementation details of the Mobilis platform. This section focuses on the evaluation of the framework especially concerning the functional requirements defined in 3. The easiest way to prove that Mobilis meets the stated requirements is to show applications that use exactly this functionality. Various mobile apps have been developed with Mobilis and there is always at least one demonstration app for each framework service. Some of these apps should be discussed in the following.

MobilisMapDraw demonstrates the server-side Collaborative Editing Service. The graphical user interface of the mobile app shows a drawing area that can be overlaid with a geographical map. The user can join a collaborative drawing session or start a new one. Within the session all participants can draw simultaneously on the map or the shared whiteboard as illustrated in Figure 7 on the left. Use cases of shared drawing can be found in various scenarios, for example in a tourism scenario to manage a trip within a group of participants.

MobilisMedia enables the exchange of multimedia content. More precisely it allows users to store their geotagged photos in repositories. The content of these repositories can be browsed by location, time and author name. The photos are arranged on a map as it can be seen in Figure 7 in the middle. Additionally it is possible to send photos directly to friends without the usage of server repositories.

The basic idea of *MobilisXHunt* is the transformation of the board game *Scotland Yard* by Ravensburger into the reality. It is a game for two to six players. One player is the wanted Mister X and the others are the agents looking for him. The position of Mister X is unknown to the agents but will be published regularly, for example in every third round. In the original game every player has various tickets (taxi, ferry, bus, tram) to move on the board game. The game ends once an agent is on the same station as Mister X or if Mister X could not get caught within a predefined number of rounds. In MobilisXHunt the players are provided with smartphones (with built in GPS receivers) and go on the hunt for Mister X in a real city using real transportation means to get from one station to the next. The user interface of MobilisXHunt is shown in Figure 7 on the right.

MobilisLocPairs is a transformation of the famous children's game Pairs. It can be played inside of buildings which are tagged with special bar codes

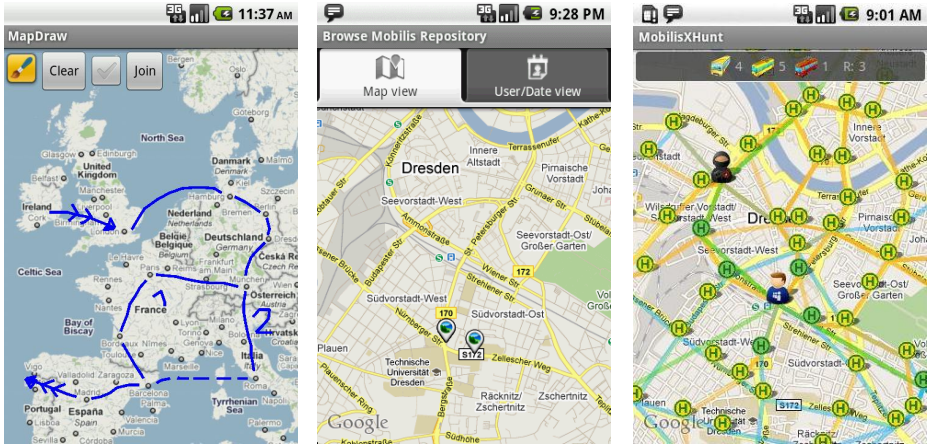

Fig. 7. Screenshots of MobilisMapDraw (left), MobilisMedia (middle) and MobilisXHunt (right).

(QR codes). Behind each code hides one card with a picture. By scanning these codes, the players can see these pictures on their smartphones. The game is also turn-based like the original Pairs but it must be played in teams of two players. Each round has a predefined maximum time (e.g. 60s). In this time the first teammate virtually turns a card by scanning the first bar code. The second teammate then has the task to find the corresponding card / bar code. The aim of the game is to find as much pairs as possible. There is also a special benefit in the game. In every bar code scanning process the smartphone measures the signal strength to all visible WiFi access points and saves this information in a fingerprint which is then sent to a central server. These fingerprints can later be used to improve the indoor positioning in the building where the game took place.

MobilisGroups is a mobile application for location-based group formation, that uses the server-side Grouping Service of the Mobilis framework. The management of user groups as it is known from common social networks is enriched by temporal and spatial aspects. Most of the groups (e.g. associations, communities of interest, sports clubs) and events (e.g. concerts, parties, BarCamps) are located at a special place, for example where the head office is located or meetings take place. These groups sometimes also have an allocated time period, that for example defines when an event happens. The idea of MobilisGroups is to extend the group management of common social networks with these temporal and spatial values as restrictions. To join or even to see certain groups the user has to be at a certain place at a certain time. The user interface of MobilisGroups is shown in Figure 8. The main component is the map view showing all visible groups in the surroundings of the user. Furthermore the user can administrate own groups, see a list of his friends and browse through the groups they are

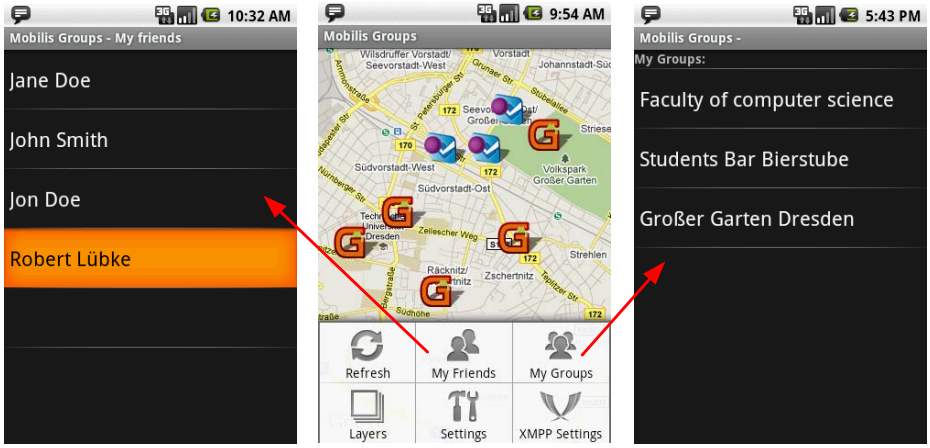

Fig. 8. Screenshots of MobilisGroups.

subscribed to. Every group also has a multi-user chat room that can be used by all members.

MobilisContext is a prototype app with the main goal to demonstrate the functionality of the server-side User Context Service. In the tabs of the application's main view one can see the different types of context information that can be shared: the mood of the user (based on XEP-107 [16]), the music he is actually listening to (based on XEP-0118 [17]) and his current location (based on XEP-0080 [15]). Besides these standardized context formats it is also possible to publish and subscribe to generic context information updates. If a user subscribes to another user's context updates, an authorization request is sent to the given XMPP-ID of the user.

MobilisBuddy realizes location-based search for friends on mobile devices. The start screen of MobilisBuddy is a normal map view on which the user can see his own position. In the menu one can organize different XMPP and Facebook accounts. These accounts are used to fetch the user's buddy lists. If one of the user's Facebook friends or XMPP buddies also uses MobilisBuddy and if this friend is located within a special visibility radius (e.g. 500m) of the user, then both receive a notification. So MobilisBuddy is a mobile application for proximity detection and proximity alerts. Within the radius both users can see each other's live location updates. That makes spontaneous meetings very easy. If one user leaves the visibility radius, one can see a gray flag on the user's last known location. This improves the privacy, because only nearby friends can receive live location updates.

This section showed that the framework functionality is used in real apps. We also wanted to point out that the developed apps come from various areas and therefore cover a wide range of use cases. Most of the presented mobile apps are prototypes of student theses or the results of practical courses for mobile app development. All student developers had enough time to get familiar with

the Mobilis framework and it's functionalities. The detailed investigation of the learning curve remains an open issue.

In a survey with 35 students who were developing mobile apps we wanted to evaluate the benefit for the developers. The majority (56,25%) of the mobile app projects included features that are offered by the Mobilis framework. The concrete benefit (e.g. in time and LOC) will be explored in future surveys.

7 Conclusion

The main idea of this paper is to provide a reusable framework for mobile social software. It contributes with an infrastructure for easy communication, collaboration, grouping, media sharing and other functionalities that are often used in mobile social networks. A unique feature is that major parts of the framework can be used independently from the mobile platform. The evaluation section showed the variety of mobile applications that were already implemented with the Mobilis framework.

The protocols, the prototype implementations and the framework components are freely available on our Sourceforge page[13] and can be reused by other applications. We are constantly refining and extending the framework and want to encourage the community to use it for their own research projects.

Acknowledgment

The authors want to thank the many contributors of the Mobilis platform for their conceptual and implementation work. The thesis of Nikolas Jansen served as a very good basis for section 5.6. Daniel Esser and the anonymous reviewers helped to improve this article with their feedback.

References

1. Clint Heyer, *Mobile Social Software.* Phd thesis, The University of Queensland, April 2008.
2. M. Endler, A. Skyrme, D. Schuster, and T. Springer, "Defining Situated Social Context for Pervasive Social Computing," in *Second IEEE Workshop on Pervasive Collaboration and Social Networking (PerCol)*, (Seattle, USA), March 2011.
3. M. Arb, M. Bader, M. Kuhn, and R. Wattenhofer, "VENETA: Serverless Friend-of-Friend Detection in Mobile Social Networking," in *4th IEEE International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)*, (Avignon, France), 2008.
4. A. Beach, M. Gartrell, S. Akkala, J. Elston, J. Kelley, K. Nishimoto, B. Ray, S. Razgulin, K. Sundaresan, B. Surendar, M. Terada, and R. Han, "WhozThat? Evolving an ecosystem for context-aware mobile social networks," *IEEE Network*, vol. 22, pp. 50–55, July 2008.

[13] http://mobilisplatform.sourceforge.net/

5. K. A. Li, T. Y. Sohn, S. Huang, and W. G. Griswold, "Peopletones: a system for the detection and notification of buddy proximity on mobile phones," in *MobiSys '08: Proceeding of the 6th international conference on Mobile systems, applications, and services*, (Breckenridge, CO, USA), 2008.
6. Andoni Lombide Carreton, Dries Harnie, Elisa Gonzalez Boix, Christophe Scholliers, Tom Van Cutsem, and Wolfgang De Meuter, "Urbiflock: An experiment in Dynamic Group Management in Pervasive Social Applications," in *First International Workshop on Communication, Collaboration and Social Networking in Pervasive Computing Environments (PerCol)*, (Mannheim, Germany), 2010.
7. C. M. Gartrell, "SocialAware: Context-Aware Multimedia Presentation via Mobile Social Networks." Master Thesis, University of Colorado, Department of Computer Science, 2008.
8. Dario Bottazzi, Rebecca Montanari, and Alessandra Toninelli, "Context-Aware Middleware for Anytime, Anywhere Social Networks," *IEEE Intelligent Systems*, vol. 22, pp. 23–32, 2007.
9. A. Gupta, A. Kalra, D. Boston, and C. Borcea, "MobiSoC: a middleware for mobile social computing applications," *Mobile Networks and Applications*, vol. 14, pp. 35–52, 2009.
10. A.-K. Pietiläinen, E. Oliver, J. LeBrun, G. Varghese, and C. Diot, "MobiClique: middleware for mobile social networking," in *WOSN '09: Proceedings of the 2nd ACM workshop on Online social networks*, (Barcelona, Spain), 2009.
11. "XMPP Standards Foundation," June 2011.
12. P. Saint Andre, "XEP-0045: Multi-User Chat," tech. rep., XMPP Standards Foundation, July 2008.
13. J. Hildebrand, P. Millard, R. Eatmon, and P. Saint Andre, "XEP-0030: Service Discovery," tech. rep., XMPP Standards Foundation, June 2008.
14. P. Millard, P. Saint-Andre, and R. Meijer, "XEP-0060: Publish-Subscribe," tech. rep., XMPP Standards Foundation, July 2010.
15. P. Saint Andre and J. Hildebrand, "XEP-0080: User Location," tech. rep., XMPP Standards Foundation, September 2009.
16. P. Saint Andre and R. Meijer, "XEP-0107: User Mood," tech. rep., XMPP Standards Foundation, October 2008.
17. P. Saint Andre, "XEP-0118: User Tune," tech. rep., XMPP Standards Foundation, January 2008.
18. Thomas Muldowney, Matthew Miller, and Ryan Eatmon, "XEP-0096: SI File Transfer," tech. rep., XMPP Standards Foundation, April 2004.
19. J. Hildebrand and P. Saint Andre, "XEP-0138: Stream Compression," tech. rep., XMPP Standards Foundation, May 2009.
20. A. K. Dey, "Understanding and Using Context," *Personal Ubiquitous Comput.*, vol. 5, pp. 4–7, January 2001.
21. R. Lübke, D. Schuster, and A. Schill, "Mobilisgroups: Location-based group formation in mobile social networks," in *2011 IEEE International Conference on Pervasive Computing and Communications Workshops (PERCOM Workshops)*, pp. 502 –507, March 2011.
22. I. Paterson, D. Smith, P. Saint Andre, and J. Moffitt, "XEP-0124: Bidirectional-streams Over Synchronous HTTP (BOSH)," tech. rep., XMPP Standards Foundation, July 2010.
23. I. Paterson and P. Saint Andre, "XEP-0206: XMPP Over BOSH," tech. rep., XMPP Standards Foundation, July 2010.