A Web Service for Flexible Integration of Mobile Applications with Social Networks

Victor Pantoja
Globo.com
Av. das Américas 700, Bloco 2
Rio de Janeiro, Brazil
+55-21-24836880
victor.pantoja@corp.globo.com

Markus Endler
Department of Informatics, PUC-Rio
Rua Marquês de São Vicente, 225
Rio de Janeiro, Brazil
+55-21-35271500 ext. 4309
endler@inf.puc-rio.br

ABSTRACT
The prospect of coupling social computing with sensing capabilities of current mobile devices makes it possible to provide social applications with higher degrees of context-awareness, detection of activities of individuals and groups, as well as implicit social interaction through sharing of context-sensitive information. In addition to the pervasive context obtained from the mobile devices, the links and user interactions in social networks can be regarded as rich sources of information for pervasive applications. In this paper we present Mobile Social Gateway (MoSoGw), a web service that provides a generic interface for optimized information transfer between mobile devices and different social networks, as well as third-party web services. Its interface is generic in that it makes transparent to the mobile client application the interaction with social networks and web services. Performance and scalability were major concerns when designing and implementing MoSoGw. Hence, its architecture and all technologies used in its development have been carefully chosen so as to scale to large numbers of clients and support high volume of concurrent requests.

Categories and Subject Descriptors
C.2.4 [Distributed Systems]: Client/server, distributed applications
C.4 [Performance of Systems]: Performance Attributes

General Terms
Performance, Experimentation

Keywords
Social Networks, Scalability, Web Service, Social Gateway

1. INTRODUCTION
Technical advances of mobile technologies and the Web have promoted new forms of social interactions relying on large networks of sites, web services and user contacts, employing different forms of communication and sharing, and where mobile devices are used to recommend, complement or replace direct, face-to-face interaction. The introduction of online social networking and Mobile Social Software (MoSoSo) made social connectivity even more explicit, offering access to functionalities built around the interconnection and matching of user profiles and contexts.

The most evident consequence of mobile communication is easier planning and execution of everyday tasks, including the communication and coordination with several other users. However, phone calls and text messaging are feasible only within a small group, and hence do not scale. In this context, Mobile Social Software represents a natural complement to the original mobile services by extending interpersonal to network interactions through the many-to-many communication paradigm of online social networks.

As the virtual and the physical worlds are rapidly growing together, online social networks are increasingly used as platform for manifold forms of interaction, collaboration and information sharing, both in synchronous and asynchronously modes. The prospect of coupling social computing with sensing capabilities of current mobile devices makes it possible to provide social applications with higher degrees of context-awareness, detection of activities of individuals and groups as well as implicit social interaction through sharing of context information. In addition to the conventional means of detecting and using pervasive user context, such as her position or movement pattern, user interactions in social networking applications can be considered as data sources for extracting information implicitly available from user’s social network graph.

However, current MoSoSo is usually developed from scratch, without support for uniform and extensible interaction with online social networks, other web services, and mobile middleware systems for context processing. Moreover, most of the systems lack means of optimized data transfer between the mobile device and the web services, and vice-versa. For example, when current mobile apps access Google Maps for retrieving some information, e.g. a travel route, a huge quantity of data is transferred, but only a small fraction of it is really required.

In this paper we present Mobile Social Gateway (MoSoGw), a web service that provides a generic interface for optimized information transfer between mobile devices and different social networks, or other web services. This interface is generic in that it makes transparent to the mobile application the interaction with third-party web services, as well as different social networks. This means that only MoSoGw must run the specific communication...
protocols of each social network, and that the service can be extended to interact with new social networks. MoSoGw also allows third-party Web services to subscribe for specific context information being generated at the mobile device (such as, type of wireless connectivity, instant messenger or call record, geographic position, etc.). This kind of integration with MoSoGw will be described in more detail in section 4.

When we designed and implemented MoSoGw, performance and scalability were major concerns. Thus, all technologies used in its development have been carefully chosen so as to scale to thousands of clients and support high volume of concurrent requests.

The remainder of this paper is organized as follows. In the next section we classify the types of context information and of mobile social networks we are considering and present a small example scenario which shows the potential uses of the combination of pervasive and social context. In section 3, we present some related work on this subject. Section 4 presents the overall architecture of MoSoGw and the main components of its current implementation. Section 5 presents two prototype applications based on MoSoGw, and section 6 shows some results of our initial performance evaluation of our implementation. Finally, in section 7 we conclude our article with some discussion about performance and scalability.

2. TYPES OF CONTEXT AND MOBILE SOCIAL NETWORKS

Most research on mobile and pervasive applications have considered four types of context [1]: the computational context, i.e. states of the mobile device’s computation and communication environment; user context, i.e. user profiles, preferences and roles; physical context, i.e. the state of the user’s environment as probed by the device’s built-in sensors, such as the position, and temporal context, i.e. time, day of the week, etc. In the following, we will call all these kinds of information the pervasive context, as it can be directly obtained from the mobile resources or sensors, and reflects the current mobile user’s situation. On the other side, every person has a social context, which consists of the group of people with whom he/she interacts, and by which means this interaction takes place. Although the entire social context of a user can never be fully grasped, data mined from online social communities (a.k.a Social Networks) certainly gives a good approximation of user’s social links. Moreover, as social networks are increasingly used through mobile phones, it becomes clear that information obtained from social networks is a very important element of the mobile user’s context, as it will probably determine much of the user’s behavior and expectations.

More recently, several research works have studied the potential benefits and challenges of merging/combining the pervasive and social context (and thus deriving the user’s Pervasive Social Context) [2], so as to explore new mobile collaboration applications. In particular, Rosi et al [3] have identified four general categories of mobile social networks according to the way of combining pervasive and social information and of deriving specific Information out of them: (a) extract information from social networks to feed pervasive applications and services, making them aware of the social links and activity; (b) social networks are used to collect and organize context data collected from the mobile device’s sensors; (c) build an application specific social network overlay, e.g. a social application that performs the integration of pervasive and social context; and (d) develop a mobile application and software infrastructure that collects social interaction data and enriches/combines it with pervasive context information. In this classification, our approach is closest to categories c and d, since pervasive social context is neither processed nor stored in the social network, nor at the mobile clients, but instead on the gateway service that interfaces several social networks and mobile clients.

In order to illustrate the potential use of the combination of pervasive and social context, we describe the following scenario of a hypothetical mobile social application, the Academic Car Sharing (ACS): Paul is professor at PUC-Rio, one of the five major universities in Rio de Janeiro which have recently created a multi-institutional social networking application for car sharing (ACS) in the city. Before leaving home, Paul launches the app on his smart phone and registers his destination, the approximate itinerary and the time he intends to leave. By giving a ride, and thus having more than just himself in the car, he is entitled to use the fast tracks in the city, which significantly reduce his trip to the university. At a nearby city district, Anna, a master student at UniRio, regularly takes a ride to her university using the ACS, where she already registered her daily trip schedule and destinations. It turns out that both Anna’s place and UniRio are more or less in the same general direction that Paul will be driving. As soon as the ACS detects a potential car sharing opportunity between Paul and Anna – i.e. because her current location is only 150 meters from an intersection on Paul’s itinerary and the expected trip duration from Paul’s place to the intersection also approximately matches the time Anna wants to leave – both Paul and Anna get a notification from ACS on their smart phone informing about the matched car ride opportunity. Then, ACS opens an IM window, and Anna and Paul exchange a few messages to inform each other about details of the meeting point, color of the car, etc. Moreover, each of them receives a system generated pass-phrase that will be used to authenticate each other at the encounter. As being part of the academic social network, ACS also informs both users that Anna’s supervisor is Paul’s colleague and is working on a similar project as his, so that they have a lot to chat during the ride. Before dropping Anna off at UniRio, they decide that it would be good to arrange a formal work meeting among the groups in the next few days.

3. RELATED WORK

Following recent research projects and systems are targeted at developing pervasive social applications: CenseMe [6] is a social network app that collects and shares client context information to classify and infer the probable current user activity. Although there is no effective integration with social networks, Common Ground [7] shares context data and derives common interests of co-located users.

SAMOA [8] provides features for managing and propagation of location based information in social networks. However, it does not obtain dynamic data from social networks, which could be used in mobile social networking.

MobiSoC [10] provides support for development and deployment of MSCAs (Mobile Social Computing Applications). It provides a platform for capturing, managing and sharing social states of physical communities and also an API. It does not use social networks to get the user context.
MobiClique [11] is a platform for mobile ad hoc networks. There is an exchange of profiles and contacts. It synchronizes contact information with Facebook but does not get real-time information about users.

MoSoGw is not aimed at a specific mobile social application, but on building a generic and flexible proxy that can interface with many sorts of social networks and can handle up and downstream flow context information for large sets of concurrent mobile clients.

4. THE MOBILE SOCIAL GATEWAY (MoSoGw)

The solution has three tiers: a mobile client, the web service (MoSoGw) and one or more social networks. Thus, clients will communicate with the social networks (or other web services), only through the MoSoGw service that will play the role of a proxy, both for the clients and for the social networks.

4.1 Overall Architecture

The MoSoGw is a web service responsible for filtering, caching and distributing pervasive context data from the mobile device to the social networks, and vice-versa, data obtained from the user’s social networks to the mobile client. In either way, the primary goals of MoSoGw are following: minimize the amount of transferred data, allow for some kind of processing of the pervasive and social context data, and provide a uniform interface to the social networks, hiding from the mobile client the details of specific procedures, data formats and protocols. Since each social network adopts specific operations and protocols, and data processing should be configurable, MoSoGw has three kinds of modules: interfaces with the social network, interfaces with the mobile client, and context data processing (e.g. filtering) or storage (e.g. caching).

As shown in Figure 1, MoSoGw obtains the social context information (i.e. the user’s contacts, recent interactions/posts, etc.) from social networks (A) through a specific interface, extracts the relevant data and forwards the filtered information to the mobile client (B). In the opposite direction, mobile client sends pervasive context data obtained from the device and/or its sensors (e.g. the user’s connectivity status, accelerometer data, geographic location) to MoSoGw (C), which caches and pre-processes the data and then distributes it to connected social networks (D) and third-party web services (F), according to the user’s distribution preferences.

We can observe that MoSoGw has a specific interface module for each social network available and also specific modules for filtering, caching and processing context data. In (D), each interface must know the communication protocol used for each social network. All the communication between applications and social networks, as seen in figure 1, will be only through MoSoGw. An important feature of MoSoGw is that through its interfaces (E, F) it enables third-party web services to process context information, transforming them into more relevant information and then perhaps sending it to social networks or to mobile clients through MoSoGw (E). Upon registration, third-party web services get a unique service ID that uniquely identifies them. At subscription, they inform what kind of context they are interested in and provide an URL to be called when a new piece of context data arrives. By doing so, this third-party service will be made available to the mobile social application users. When a new context data arrives, all third-party web services which subscribed to this context type receive a HTTP POST through the informed URL. Alternatively, applications can actively obtain any context data by requesting them synchronously through an API.

![Figure 1. Integration of mobile devices with social networks and third-party Web Services using MoSoGw.](image)

On the mobile client, pervasive context data is obtained using the Context Management Service (CMS), a framework that facilitates the development and dynamic deployment of Context providing modules (CxP) [4]. CMS also supports the implementation of Context Providers of higher-level context information that combine, aggregate or process basic context data from other CxPs into higher-level context information (e.g. Δlocation / Δtime = velocity). Thus, it is possible to establish a graph of CxPs, where some CxPs query/subscribe to context data updates from other CxPs. Also applications can query or subscribe to context data updates from any CxP. It is also possible to define Context Providers that obtain their data from a remote source, such as social information from an external web service.

The application presented in section 2 could be described in terms of the proposed architecture for the MoSoGw. In this case, the ACS could send the context related to the position and speed to MoSoGw (G, H and C), which would feed it into the Academic social network (D). In the other direction, information from the social network, e.g. matching interests of users that have a direct or indirect social link, would be extracted and delivered to the mobile client (A) so that Anna can decide if she will opt for this ride in Paul’s car. Alternatively, some pervasive or social context information could be delivered to a third-party web service application (F), which could process this information to detect, correlate data and infer other information such as the traffic condition on the way to Anna’s university. After processing, this information would be sent back (E), and either forwarded to the corresponding mobile clients, or posted to the social network using the interfaces already available in MoSoGw.

Although this example is geared towards the particular context ‘user location’, MoSoGw is designed to be generic with regard to the supported context types, and new context types can be added at any time.
4.2 Application Interface
The existing features of MoSoGw are available through two APIs, allowing client applications and third-party web services to be built by using any platform/environment that supports the JSON format (used in response messages from the server). The first API is provided directly by MoSoGw and is composed of several REST services. In this case, applications must implement HTTP requests and know MoSoGw interfaces and URLs.

The second is an Android Java API, which is better suited to mobile app developers, since HTTP and JSON details are hidden behind the Java class abstraction, i.e. programmers simply need to instantiate a class, and get access to all MoSoGw features. This API is used by prototype applications described in section 5.

In both cases, every time a new social network or web service is added, it is automatically recognized by the API. Hence, application developers need only be aware of this dynamicity. Most development complexity related interaction and discovery protocols will be handled at the MoSoGw side.

4.3 Current Implementation
Our current version of the Mobile Social Gateway is implemented in Python. Each component in the software architecture of MoSoGw was chosen in order to achieve high performance and scale to thousands of simultaneous requests. MoSoGw uses the nginx HTTP server [5] which is known to be very fast and use few resources such as CPU and memory. Unlike traditional servers, nginx does not use threads to make connections but uses an asynchronous, event-driven architecture (Asynchronous non-blocking I/O), that makes it more scalable than other HTTP servers, and enables nginx to serve thousands of concurrent clients. Another important component is Tornado, a lean Python framework. Its main features are: simplicity, high performance, open source code, non-blocking I/O and low resource consumption. Regarding connection to database server, we chose SQLAlchemy [13].

Integrating pervasive context information flows with social networks can be computationally expensive due to data aggregation, filtering or pre-processing needs. Thus, to achieve higher performance, we decided to separate the distribution of context information to social networks into two steps: first, the context data sent by a mobile client is queued by the Tornado application with additional information (e.g. which social network should the context data be sent to). Second, a standalone process processes this queue and, for each entry, posts the context data to a template file, which is simply an html file with some specific markup that will be pre-processed (together with all data) by a template manager (i.e. Mako [16]). In Java, for example, templates are represented by JSP’s.

Mobile Social Gateway system has a modular design and a scalable architecture suited to execute in a cloud or cluster environment: an instance of nginx distributes the load to multiple instances of Tornado. The number of instances depends on the server settings and is proportional to the number of processor cores. Besides these two components, Mobile Social Gateway uses a MySQL database to store contexts, users and friendships information, and invites some other information. We named the set nginx plus n Tornado instances a "box", shown in figure 2.

The figure 2 depicts only one instance of memcached [15] and a database but there could be more instances, e.g. on any node of a cluster. If we take a closer look at this architecture and examine only the core MoSoGw service, it will consist of the following parts, depicted in Figure 3.

In figure 3, a Tornado instance receives the request forwarded by nginx and sends it to the specific handler, which can call a repository object directly or render a template. Repository classes seek data in the cache and if not found, access the database for the data. The response can be a simple json or something more complex, like an html file. In this last case, handler will pass all data to a template file, which is simply an html file with some specific markup that will be pre-processed (together with all data) by a template manager (i.e. Mako [16]). In Java, for example, templates are represented by JSP’s.

4.4 Cache Architecture
To prevent that client requests arrive at the database or even at the python application, we implemented all possible cache levels offered by the chosen technologies: disk cache (nginx), memory cache (memcached) and a SQLAlchemy session cache.

4.4.1 Nginx
First layer of cache, nginx can be set to save to disk the answer to a given request. You can define different areas of cache, each one with a specific configuration of expiration time and maximum size. Nginx will generate a hash based on the requested URL and the answer will be saved in a directory defined in
proxy_cache_path. So, next time a same request is made, this cache will be used. In order to support user-specific caching, user and application IDs will be used to generate cache indexes, (i.e. nginx’s proxy_cache_key).

4.4.2 Memcached
Unlike nginx, memcached uses memory and it is the second layer of the Mobile Social Gateway cache architecture, responsible for storing answers to database requests. It takes a pair (<key>, <value>) and stores it in memory. As an example, we have a pair that stores information about de logged user. The key is a token generated when user authenticated and the value is the object user. So, when a token is received, it’s possible to retrieve which user is logged. To reduce the used memory space, instead of storing the result of a request in the database, MoSoGw stores the list of ids returned, and for each id, stores the returned object. For example, if a user requests a list of all his friends, MoSoGw saves in memcached only the list of ids found, and from each id, the correspondent object will also be saved in memcached. Thus, if a subsequent request looks for a specific object of type friend, then this object is already in memory. Hence, the use of memcached significantly increases the system performance. If the list of objects (not ids) were stored, then for any change on the object it would be necessary to invalidate the entire list. Hence, the object update will reflect on all services that share the object.

4.4.3 SQLAlchemy
SQLAlchemy performs session caching of any created object. For example, when an object of type User is created, firstly it is materialized only in a SQLAlchemy session, and database insertion/update is deferred to the termination of all operations on this object. By such, all following request to this object it can be directly retrieved from the session cache, speeding up the response time. SQLAlchemy also guarantees the object’s consistency by implementing a validity flag of the object, i.e., if someone changes the object, this change flag will be set and the object will be re-inserted into the database and also updated in the session cache.

5. MoSoGw Prototype Applications
In order to demonstrate the usefulness of MoSoGw, we have developed two prototype applications.

5.1 Mobile Social Share
Mobile Social Share (MSS) is an application that allows mobile users to share context information about their current movement patterns. It uses the geographic position and velocity of several users, provided by their mobile device’s sensors, to estimate the traffic conditions in different parts of a city. Additionally, the users can use a select box of the GUI to manually input their context information (i.e. slow, normal, or fast movement), or write some arbitrary text about the current traffic condition that is to be shared with their acquaintances in the social networks. The information is then either posted to all available social networks - so far we have implemented interfaces to Twitter and Facebook - or to some particular social network selected by the user. Moreover, at any time the user can enable and disable the automatic sending of this context information to social networks. Thus, Mobile Social Share exercises the flow of information from the mobile client into the social network.

5.2 ActiveCal
ActiveCal is a mobile social app used for monitoring participants of a scheduled meeting. For each calendar entry that is a meeting, each user of ActiveCal is able to learn about all participants’ current position, remaining distance to the meeting point and estimated time of arrival at the meeting point. ActiveCal integrates with Google Maps API and Google Calendar API. The first API provides the distance between two geographic points and the possible routes to the destination. The second one provides meeting information, such as date, time and location. Contrary to the previous application, ActiveCal does not distribute context information through social networks, but evaluates MoSoGw’s ability to optimize the transfer of data from the Google web services to the mobile clients. For example, the Maps API per se provides a large amount of largely useless/redundant information that is filtered out by a MoSoGw RouteExtractFilter, so as to forward only the relevant route distance and time information to the ActiveCal client.

Both applications use the Context Management Service (CMS framework) [4], an Android service that manages context providing components, which either probe sensor data of the mobile device, or aggregate/process higher-level context data. Within CMS, each type of context data is obtained or produced by a specific Context Provider (CxP), that CMS may deploy individually but which may depend on other, lower-level CxPs. CMS also supports the discovery and dynamic download of new Context Providers from a remote Repository of CxPs. In addition to the already available “basic” CxPs (e.g. GPS position, connectivity type, time/date, remaining energy, etc.), we developed following specific context providers (CxPs) of higher-level context information, as support for the Mobile Social Share and ActiveCal applications:

Distance Context Provider: these CxPs inform the user’s current location (GPS- or cell-based) and velocity (in km/h and semantically: slow, normal, fast), respectively.

Meeting Context Provider: this CxP accesses Google Calendar to retrieve information about scheduled events, such as time, location and list of participants.

6. PERFORMANCE EVALUATION
In order to evaluate the performance of MoSoGw, we applied some load tests using LoadStorm [12]. Tests where done using a VPS with 512MB RAM, running Ubuntu 10.04. We tested a simple MoSoGw box with one nginx instance communicating with a Tornado instance using a single MySQL server and memcached. All the three cache layers described in section 4 were implemented.

The scenario chosen included the following steps:
1 - Create a new user
2 - Log in with a user
3 - Send an invitation to another user
4 - Send context information to Twitter though Beanstalk

We started the test with 5 simultaneous clients and then added new clients linearly up to 500 simultaneous clients in a total of 20
minutes of test. The results are show in figure 4 and 5. The x-axis is in minutes and represents time passed since test started.

In Figure 4, it can be observed that at point of peak request load (22 req/s), MoSoGw was still responding well and the throughput was proportional to the growth in the number of clients.

As shown in table 1, the average response time was measured 0.177 seconds, which is quite good. However, Figure 5 shows a peak of 4.45s in response time. Although this may not be acceptable for some specific real-time context processing applications, we believe that for most user-operated mobile social apps it is quite reasonable to assume that users can wait 4.45s to get a context information update (e.g. a location update of a friend would be delayed by this time).

The throughput drop after 20 minutes, is not relevant, as it coincides with the moment that LoadStorm concludes the test session.

### Table 1. Performance Evaluation.

<table>
<thead>
<tr>
<th>Response (avg.)</th>
<th>Requests (req)</th>
<th>req/s (avg.)</th>
<th>req/s (peak)</th>
<th>Throughput (avg.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.177s</td>
<td>14,810</td>
<td>12</td>
<td>22</td>
<td>6 Kb/s</td>
</tr>
</tbody>
</table>
7. DISCUSSION AND CONCLUSION

In this paper we presented a Mobile Social Gateway (MoSoGw), a prototype of web service that aims at supporting the development of mobile social applications by acting as a proxy between mobile clients and social networks. It is used for filtering, caching and distributing pervasive context data from the mobile devices to social networks, and vice-versa, for obtaining stripped-down social context information and delivering it to the corresponding mobile user devices.

MoSoGw does not have any built-in mechanism for processing or matching users’ context information and social network data. However, it was conceived as a generic gateway to enable third-party web services to perform such work. Actually, many types of context and social data filtering and correlation services can be implemented by other applications or web services. All they need to do is subscribe to context data managed by MoSoGw.

Our initial usage and performance tests are non-conclusive, and we are aware of the need to develop more mobile social applications based on MoSoGw and to test it in many more workload scenarios. However, we believe that our prototype MoSoGw already has some promising architectural features. For example, in this paper we have tested only a single instance of MoSoGw, i.e., only one box, cf. Figure 2. However, sets of boxes could be configured and deployed on dozens of virtual machines in a cloud, which would yield much better performance results in terms of throughput and response time. Moreover, memcached can also be distributed on several machines, so that request workload can be automatically distributed among all instances. Thus, as one of our next steps we will test a parallel/distributed deployment MoSoGw and precisely assess its scalability. In another line of future work, we will implement interfaces to external web services for performing context data analysis, and correlation and intelligent matches of pervasive and social context data.

8. REFERENCES


[12] LoadStorm: http://loadstorm.com


