How applications become social

Georgi Kikilingovski, *Computer Science, TU Berlin*

Abstract—It was very interesting to watch how Googles OpenSocial was published and how it was received by the public. It is been five years now and in the beginning everybody was expecting a new Facebook. Today, most people know that Google+ is their answer. But still the majority is not sure what OpenSocial really is. It is a set of common application programming interfaces (API) for web-based social network applications. The OpenSocial API is based entirely on JavaScript. With HTML, XML and JavaScript skills it is possible to use OpenSocial to extend your web application or web page into an OpenSocial Application. Also OAuth 2.0 and OpenID are supported for a secure open authentication and authorization.

So OpenSocial primarily targets on developers, who are interested in building up applications for social networks or in connecting their content and deploying it on the internet.

Index Terms—OpenSocial, Open Graph Protocol, Social Graph, Shindig, REST

1 Introduction

OpenSocial enables any web page to become a rich object in a social graph. It is a public specification that defines two components. One is the hosting environment called container and second is a set of common API for web-based applications. It was released November 1, 2007 and was initially developed by Google, MySpace and some other partners[1]. In the beginning it was primary conceived for social network applications. Nowadays it has become adopted as a general use runtime environment for allowing untrusted and partially trusted components from third parties to run in an existing web application. Besides this progress, the OpenSocial Foundation also integrates numerous other open web technologies like OpenID and OAuth 2.0. Since OpenSocial 2.0 also , Activity Streams and protable contacts are supported as well. A hosted social application which can run as a desktop, browser or mobile app and is using the OpenSocial API will be interoperable with any other social network that supports them (see fig.1), including features on sites such as MySpace, Orkut, Friendster, Yahoo!, Xing and many more[2].
The component which defines the API can be divided into three core parts:

- user profile - enables accessing and sharing personal data
- friends - allows to view the social graph from a user's sight and to trigger relationships queries
- activities - allows text-based communication and notification between users

Fig. 1. OpenSocial overview

Containers are a non proprietary further development of Google Gadgets[1]. They define the style and manage the crawled social data in

• *G. Kikilingovski is with the Department of Telecommunication Systems, TU Berlin, Berlin.*
E-mail: georgi.kikilingovski@campus.tu-berlin.de

Manuscript received June 19, 2012.

1. Google Gadgets are dynamic web content items that can be embedded on a web page

web browsers. Containers use JavaScript interfaces or the OpenSocial-REST protocol to get access to the OpenSocial API.

1.1 Social Application Container

In principle a social application (social app) provides a social network site with extra functionality for the user. Every social app can hold own set of some additional features. Theses social apps are hosted on web pages in so called containers[3, S.2].

As briefly described in the summary a container consists at least of three core components:

- user profile
- friends and connections
- activity stream

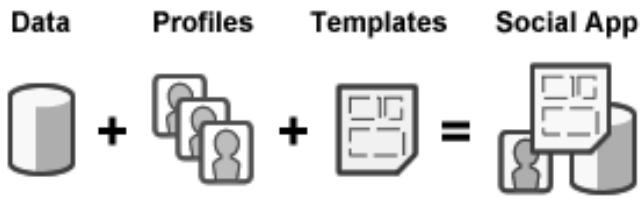

Fig. 2. Social App components

All personal information about the user like sex, age, interests, favorites, etc. are stored in the *user profile*. The data can be shared with specific friends, groups or even the whole world, it depends on the privacy settings. With this data it is possible to develop highly targeted and personalized social apps. More data involves better user experience in the personalized way. Many containers provide statics about how complete a profile is and prompt users to complete it. Through this mechanism social apps become more personal.

Information about a user's social graph and her relationships to others can be found in the *friends and connections* component. It is possible to set queries from the user's view on the social graph and get information about her ties. The basis of any social graph are the relationships between users. Users want to map their relationships to family, friends, coworkers and all the other persons with any social relevance online. Grouping these relationships is important for later targeting methods e.g. to promote or recommend new features or other applications. This is a major feature for business use.

Activity stream is the main component in every social network. It is a text-based information and notification system. A user can use this news feed to get a collection of her friends activities. It is also in use to communicate with friends and others in the social network. Additional to that it is possible to share multimedia data like pictures and videos.

1.1.1 Container environment

Before building up a social app it is necessary to recall the differences between a traditional application and a social container environment. The social container environment is a black box and consists of three layers and each layer may contain same processing, filtering and serving mechanisms.

Social	Traditional
Application	Application
Container Processor	Application Servers
Application Servers	

The major difference is the middle tier of the social container environment. The container defines the infrastructure and developing features. So the *application server* has to provide code first to the *container processor*, where it will be filtered, processed and served to the *application* layer.

This brings with it some advantages but also some disadvantages[3, S.8].:

+ Allows server independent developing
+ Security features are often automatically included in containers
- Container upgrades can reveal new bugs in a black box environment
- Additional container uptime is necessary for accessibility of apps

1.1.2 Application views

With views an application is able to interact with a user on a social container. An OpenSocial app can have different views, depending on where to find it on the site. All views fall into one of these groups:

- Small view - view which is restricted in size and functionality
- Large view - view which supports complete functionality for the full user experience

The most used view is the *canvas view* (large) and can be found in any application. The *profile view* (small) is a specific view which has to be installed or enabled by the user on his profile. Having a profile view is optional for an app but nice to have. There also exists the *home view* (small) that provides personal aggregated content only to the user. This specific view is only available for the user and not accessible for his friends[3, S.13].

1.2 Open Graph protocol

The Open Graph protocol (OGP) is an open API to semantically classify content of a web page so applications like Facebook can readout and utilize them. With OGP any web page can become a rich object in a social graph and access all features of any application.[ogp.me]

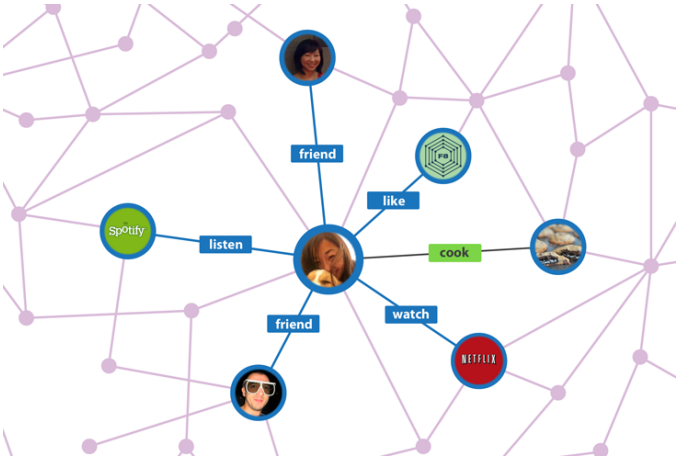

Fig. 3. Semantically clustering of web content

1.3 OpenSocial REST protocol

A client application can view and publish a user's social data via the OpenSocial REST protocol, which is based on REST[2]. With this protocol a client application can request data items (such as Activity, User Profile, or AppData items), create new items, edit or delete existing items, and filter or modify results using query parameters[4].

This protocol allows a developer to retrieve pieces of information about a user and her friends by sending HTTP GET requests to a server. For the other direction it is possible to send information with the HTTP POST request.

1.4 Shindig

Shindig is an open source project. It started in December 2007 to provide a reference implementation for the OpenSocial standard. The software contains both server-side and client-side code. Once the project is mature, an installation of this product will be capable of rendering OpenSocial applications in a web browser[5].

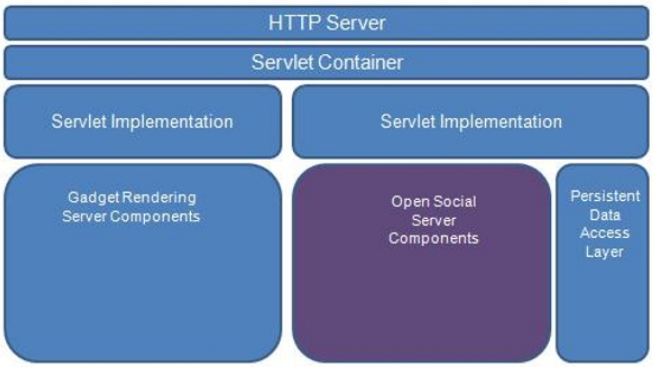

Fig. 4. Apache Shindig Java Server Side container

2 The Open Social Graph

Relationships between people is the core of a user's social graph. The links in this graph are the best sources for information about a person, his hobbies, his preferences, his habits and many other details. To engage a new user base it is important to have a relevant graph.

2. REpresentational State Transfer (REST) is a style of software architecture for distributed systems such as the World Wide Web

On the social web a user interacts with other people like friends, family, coworkers and all the other persons with any social relevance. These social links between them are referred to as direct relationships which have the highest social relevance degree. Indirect relationships describe the social links between the user's friend and the friend friends. Theses connections have the lowest social relevance degree, but their relevance can be raised by common association. Relationships can be categorized in relevance tiers[3, S.40]:

1. tier - direct relationships
2. tier - first level indirect relationship
3. tier - second level indirect relationship
n. tier - ...

2.1 User clustering

Am important mechanism for a successful application is the possibility to cluster relationships. Like in real life users will cluster their relationships. This brings a benefit for the application as well. Clusters hold additional information about users and their friends.

But on the other hand user clustering involves the challenges of *security and privacy*. These two issues cannot be handled separately. A social app has to define its limits and ensure. It would be a critical failure when during interaction the news feed would be delivered to the wrong cluster or person. Also it is important to know that clusters and relationships can change after every interaction. Correct reactions on these transitions is necessary.

2.2 Data sharing

Most people share a lot of their personal data with socials apps or containers. Two different models exist about how the data will be shared. It does not matter if it is in a social network (e.g. Facebook, Google+, Orkut) or a service (e.g. Foursquare, Gowalla):

- opt-in sharing model
- opt-out sharing model

In an *opt-in share model* it is set by default that any application or network will not share user data. The user has to enable the sharing feature manually. This model is a benefit for the user. The user will be to setup his sharing setting manually or the application should prompt him to do that.

The opt-out share model represents the absolute opposite to the opt-in one. By default any user of the service shares everything. Now the benefit lays on the application side. Most people will never bother with setup privacy or sharing settings so they ignore that. Using this model the app can instantly capture and share all of the desired information[3, S.44].

2.3 Relationship Models

Another core component in the social graph is the relationship model. Actually three basic types of relationship models exist on all the different networks or services[3, S.44]:

- Follower Model - user interacts with many others at once
- Connection Model - user interacts with just one person at a time
- Group Model - user interacts with small groups of people

In the *follower model* the user just interacts with her followers. The best example is Twitter. If a user posts a message, he can only reach his followers or when supported and set up the world. The biggest disadvantage of this model for the user is, that where are only few privacy settings and even those are limited. The only way to protect your own messages is to force followers to get a read permission. But the user is not able to manage or affect her social circle or whom she broadcasts her messages to. Social relationships formed here tend to be limited. People that are using Twitter are interested more in the content of the message than in the author. In this model it is difficult to get the same depth of social relationships like in other more relationship-centric models like the connection model. Also it is important to keep in mind that actually people can read messages which are not designated to them.

The *connection model* has a completely different approach, compared to the *follower model*.

The focus here is on the personal, specific interaction between users. The possibility to share social experiences rather than to broadcast to a large group and reach as many people as possible stays in the foreground. Users of this model tend to share many events of their lives via a network like Facebook. They exchange nearly everything with their friends. This model gives users complex settings to setup their privacy, so they can define who can read which message. Additional to that users can hide their whole profile from other users outside their social circle. The biggest challenge for this model is to design a good concept to secure social data but also share specific information.

The *group model* has similarities to the connection model but aims on the interaction and sharing experiences of groups. A group consists at many people with similar interests, backgrounds or situations. In a social network groups are so called *micrographs* which are just small portions of the entire social graph. There may be one or many groups in a graph and friends can appear in one or more groups.

2.4 Entities

The concept of user relationships in a social graph is not as easy and there exists more than one approach. But often an user may interact with many more common interest sources on the web, moving far beyond a normal relationship cluster. An approach to tackle this issue is the concept of *entity relationships*. Entities consist of any link to a user's behaviors like searches, websites visits or online purchases. By mapping a user to these entities to cluster him additionally, the traditional model of a social graph will be extend into a rich complex online personality[3, S.50].

The biggest advantage of this approach is that an application can collect much more data from the user's behaviors and use them to design a extremely personalized user experience. But in this approach it is very difficult to securely handle the user's privacy. Also it can stress current laws.

3 JUST A SOCIAL GADGET

As described in this paper a web page that hosts an OpenSocial application is called an OpenSocial container. On these containers, an OpenSocial application can run. Every Google Gadget or in this case an OpenSocial Gadget is defined by a XML construct. Listing 1 shows the content of that XML definition.

With the row...

```
Require feature="opensocial-0.8"/>
```

...a Google Gadgets becomes an OpenSocial Gadget.

Listing 1. Gadget Structure

```
<?xml version="1.0" encoding="UTF-8" ?>
<Module>
 <ModulePrefs title="TransferApp">
   <Require feature="opensocial-0.8"/>
 </ModulePrefs>
 <Content type="html">
 <![CDATA[
   <script type="text/javascript">
   </script>
   <div id="message"> </div>
 ]]>
 </Content>
</Module>
```

Currently, the gadget has no functionality. In this example the gadget should be able to request or send some data. So it is necessary to write two JavaScript-functions which can handle this. The next listing 2 is currently empty but it gives a good overview about the inner structure of a gadget.

Listing 2. Social App functionality

```
<script type="text/javascript">
  function request() {
        // some func...
  };
  function response(data) {
        // some func...
  };
  // Execute the request function when
  // the application is finished loading
  gadgets.util.
        registerOnLoadHandler(request);
</script>
```

Once the functionality is implemented, the application views have to be designed. That code has to be pasted into XML in the *CDATA* section (see listing 1). This is just a small extract of an application based on a tutorial example from http://docs.opensocial.org.

4 Conclusion

With OpenSocial a developer can quickly and easily extend any web page to a rich object in a social graph. An important advantage of OpenSocial is that it is open and also supports other open Web Standards like OAuth or OpenID. With this support the API handles most security issues in authentication and authorization at its own. The biggest advantage of OpenSocial definitely is the structure of the social gadgets/ apps. All developers for Google Gadget can start immediately developing without having to learn something new. But more important is the inner structure of the social apps. An app runs on a social container which is hosted on a web page. So it does not matter how the host is configured or which requirements are needed as long the container works correctly. This also involves some disadvantages though. A container upgrade can make your app source code incompatible or an app is not running because the container is down. Sure it is easy to get all the social data from any networks or services to use them to create a highly personalized app. But before that it is a developers duty to handle all the privacy and security challenges. Also it is important to know which kind of relationships model is needed and so on.

References

[1] *Google Launches OpenSocial to Spread Social Applications Across the Web* Retrieved 11-06-2012 from http://www.google.com/intl/en/press/pressrel/opensocial.html

[2] *MySpace and Google Join Forces to Launch Open Platform for Social Application Development* Retrieved 11-06-2012 from http://www.google.com/intl/en/press/pressrel/myspace_opensocial.html

[3] Jonathan LeBlanc, *Programming Social Applications*, 1st ed. Sebastopol, United States of America: O'Reilly, 2011.

[4] *Shindig Overview* Retrieved 11-06-2012 from http://de.wikipedia.org/wiki/OpenSocial

[5] *Shindig Overview* Retrieved 11-06-2012 from http://shindig.apache.org/overview.html