



Diagramming the Social Graph

Implementation notes on an application that visualizes the
Google Social Graph API.

Author: Tom Martin



Summary:

Google Social Graph API

Google have created a dataset with an API¹ that describes the relationships between people based on their presence on social networking sites, blogs and other social media on the web. The data is currently harvested by the Google web crawler from XFN (XHTML Friends Network²) and FOAF (Friend of a Friend³) mark up placed in web pages. XFN and FOAF are both machine-readable formats that describe the topology of a social network. XFN is a way of describing Hyperlinks in a web page whilst FOAF is an RDF document that describes the network implied by a whole site or a subset of its web pages.

Diagramming the output of this API using Altio⁴ allows a user to visually browse their social graph and better understand its structure and integrity. Users can find valuable connections with other people and sites they would not otherwise have found simply by expanding the visualization to two degrees of separation and beyond. They can also easier detect pollution and flaws in their graph; users with excessive friends / relationships and missing or erroneous connections caused by poor mark up. It also makes clearer which sites have embraced these description formats, which sites are inadequately described and which are closed systems that don't make this user data publicly available.

Architecture:

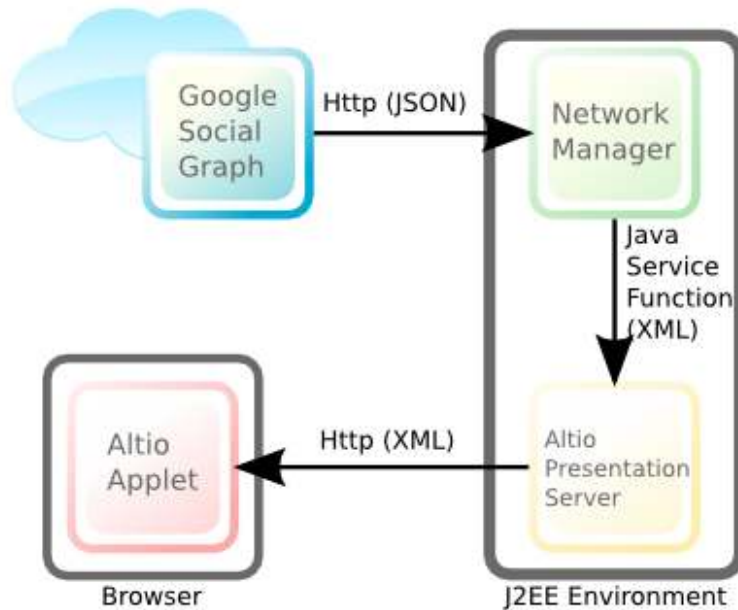


Figure 1 Architecture of application

The architecture of the application is common to many Altio based application. The Altio client applet makes a service function request, via the Altio presentation server to some business logic, in this case the Network Manager, which presents the data as a query to some data source, in this case the Google Social Graph.

Implementation:

Network Manager

The Network Manager is a custom Java class on the class path of the J2EE server that receives a request for data from the Altio client via the Altio Presentation Server in the form of a URL. It then retrieves the social graph links from Google via a REST request. The parameters are the requested URL, an “edo” parameter, set to 1 and an “fme” parameter, also set to 1.

Example Request to Google Social Graph API:

<http://socialgraph.apis.google.com/lookup?q=&fme=1&edo=1>

The “edo” parameter requests that Google returns outgoing links from the given URL. In this case we are looking for friend / acquaintance relationships. The “fme” parameter requests that all known “me” links (links that the page at this URL also claims to own)

are also returned with they're outbound relationships as well. The result is that Google searches the entire tree of "me" links beginning at the URL supplied and returns all known "friends" of each of these pages.

On receiving this data, in JSON format, the Network Manager parses it into a set of Java objects (using JSON-lib⁵ see Technical Issues) and iterates over the list of friends. Within this set of "friend" URLs there will likely be links that represent the same person in Google. As we are attempting to represent people in the graph rather than web pages we need to attempt to make further "fme" requests for each link to discover which of these links relate to the same person. Once all the links are successfully grouped the dataset is serialized into XML and returned to the Presentation Server for forwarding to the client. The abbreviated data set shown below would yield the network diagram in figure 2 and was constructed from the initial URL <http://pownce.com/jimcro>.

Example response from Network Manager

```
<PEOPLE>
  <PERSON ID="http://pownce.com/jimcro" NAME="jimcro / raindog">
    <PROFILE URL="http://pownce.com/jimcro" LOADED="Y">
      <FRIEND URL="http://pownce.com/frozonecold"/>
      <FRIEND URL="http://pownce.com/heychinaski"/>
      ...
    </PROFILE>
    <PROFILE URL="http://del.icio.us/jimcro" LOADED="Y"/>
    <PROFILE URL="http://www.flickr.com/people/raindog/" LOADED="Y"/>
  >
  ...
</PERSON>

  <PERSON ID="http://digg.com/users/shinyhappydan/" NAME="shinyhappy
dan / Dan_Bell">
    <PROFILE URL="http://digg.com/users/shinyhappydan/" LOADED="N"/>
    <PROFILE URL="http://myspace.com/shinyhappydan" LOADED="N"/>
```

```

    ...

</PERSON>

<PERSON ID="http://digg.com/users/heychinaski/" NAME="heychinaski
/ theheychinaskis / Tom_Martin / powncified">

    <PROFILE URL="http://digg.com/users/heychinaski/" LOADED="N"/>
    <PROFILE URL="http://myspace.com/HeyChinaski" LOADED="N"/>

    ...

</PERSON>

</PEOPLE>

```

Altio Applet:

Example dataset as it appears on the client

```

<graph>

<PERSON ID="http://pownce.com/jimcro" NAME="jimcro / raindog">
    <PROFILE URL="http://pownce.com/jimcro" LOADED="Y">
        <FRIEND URL="http://pownce.com/frozonecold"/>
        <FRIEND URL="http://pownce.com/heychinaski"/>
        ...
    </PROFILE>
    <PROFILE URL="http://del.icio.us/jimcro" LOADED="Y"/>
    <PROFILE URL="http://www.flickr.com/people/raindog/"
LOADED="Y"/>
    ...
</PERSON>

<PERSON ID="http://digg.com/users/shinyhappydan/"
NAME="shinyhappydan / Dan_Bell">
    <PROFILE URL="http://digg.com/users/shinyhappydan/" LOADED="N"/>

```

```

<PROFILE URL="http://myspace.com/shinyhappydan" LOADED="N"/>
    ...
</PERSON>

<PERSON ID="http://digg.com/users/heychinaski/" NAME="heychinaski
/ theheychinaskis / Tom_Martin / powncified">
    <PROFILE URL="http://digg.com/users/heychinaski/" LOADED="N"/>
    <PROFILE URL="http://myspace.com/HeyChinaski" LOADED="N"/>
    ...
</PERSON>
</graph>

```

The client is a typical Altio application developed using the Altio Designer UI builder. The diagramming is achieved using the Altio Graph Control, an extension to the current Altio tool set implemented as an Altio Custom Control. It is configured for a particular dataset, like most Altio controls, by specifying XPath statements. A subcomponent of the Graph, the Node, is created and named “Person”. The Person node in the graph is specified by the Xpath “/graph/PERSON” in its “Data Source” property so that each instance of the PERSON element under the graph element will cause a node to be drawn on the graph.

The relationships, or edges, drawn between the nodes on the graph are more complex. The key configuration properties for another subcomponent of the Graph control, a Link, are shown below.

Example configuration for a Link

Property Name	Value
Data Source	/graph//PERSON//PROFILE/FRIEND
Node A Type	Person
Node A Foreign Key	ancestor::PERSON/@ID
Node A Primary Key	@ID
Node B Type	Person
Node B Foreign Key	@personId
Node B Primary Key	@ID

The Data Source will cause a new link (or line) to appear on the graph for every instance of a FRIEND element in the data set but we must configure which nodes each link appears between. First we specify the type of node at either end of the link in the “Node

A Type” and “Node B Type” properties; in this case the Person node we defined above. The attribute that uniquely identifies each person node is it's ID attribute so we place that in the Primary Key fields. Primary keys are specified relative to the nodes data element (PERSON). The foreign keys are specified relative to the Link's data element (FRIEND). The Node A element that represents the Person at one end of the Link is it's PERSON ancestor so we use the XPath ancestor axis “ancestor::PERSON/@ID”. The other node is specified by the personID attribute on the FRIEND element so we put this in the “Node B Foreign Key” property.

Technical Issues:

Parsing the API's JSON data in Java

REST-based web services and APIs are an important element in the development of web applications and due to the prevalence of JavaScript within this space JSON is becoming an increasingly popular data format for them. Google are somewhat of an advocate for Javascript as the development language for the web and for this reason some of their API's are only available in JSON format, the social graph included. Altio currently only accepts XML as a data source but a simple solution is available, JSON can be easily parsed into a tree of Java objects using `Json-lib` ⁵. It can even be directly serialized into XML in trivial cases, however currently only when the the JSON objects have key names that make up valid XML element identifiers. Parsing the JSON retrieved from the Social Graph was a simple as calling the static `toJsonSON` method on the `net.sf.json.JSONSerializer` class.

```
JSON json = JSONSerializer.toJsonSON(jsonString);
```

Improving performance

Retrieving all the data from for even a single URL can take some time as each request from the Altio client for a URL's profile and friends could result in tens or even hundreds of requests to the Social Graph API as the “me” links for each friend are loaded. The initial implementation of the Network Manager involved making each of these requests sequentially, one at a time. A vast performance improvement was achieved by threading these requests, performing them concurrently and then processing them into the XML response once all the requests had completed.

Whilst there is little concern of surpassing Google's request query limit of 50,000 queries per user per day further performance improvements could be achieved by caching the results from the Social Graph API. This is because, whilst a user explores a social graph, they will find many of their friends have friends in common which in turn causes many duplicate requests to the API. Redundancy caused by caching is of little

concern here given that the result set for a given URL will only change as often as the Google Crawler indexes that page. For this reason we allowed a whole day as the lifespan for each cached result.

The Application

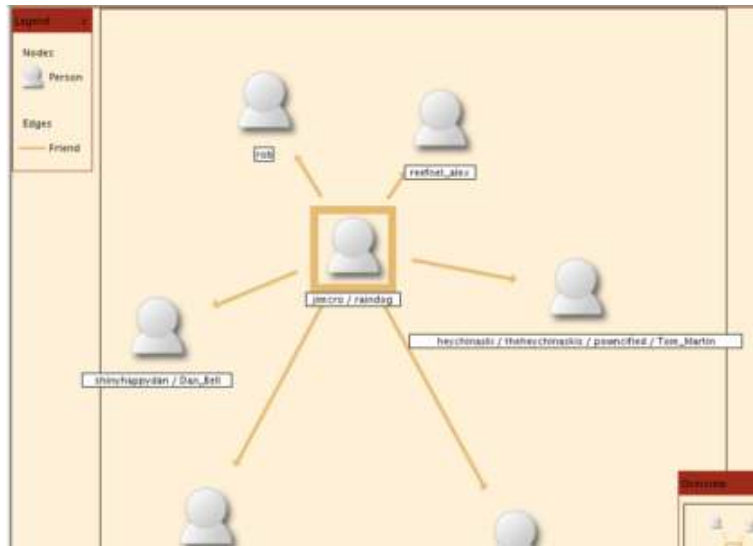


Figure 2. Small initial graph.

The resulting Altio applet client allows a user to enter the URL of one of their social network profiles or websites and the application will retrieve all profiles known by Google to belong to that person. It will provide a user with a list of their profiles (figure 4) and diagram their social network based on these found profiles to one degree of separation (figure 2). Other people, friends of the user will be displayed in the diagram as a sum of all of their discovered profiles which in turn can be queried to expand the data set in the diagram (figures 3 and 6). The richness of the Altio Graph Control within the Altio Applet allows the user to zoom and drag the graph, toggle labels and move and rearrange the nodes automatically and with the mouse. When the graph is particularly busy, the labels can be hidden to improve clarity and users can be found by searching on user name (figure 5).

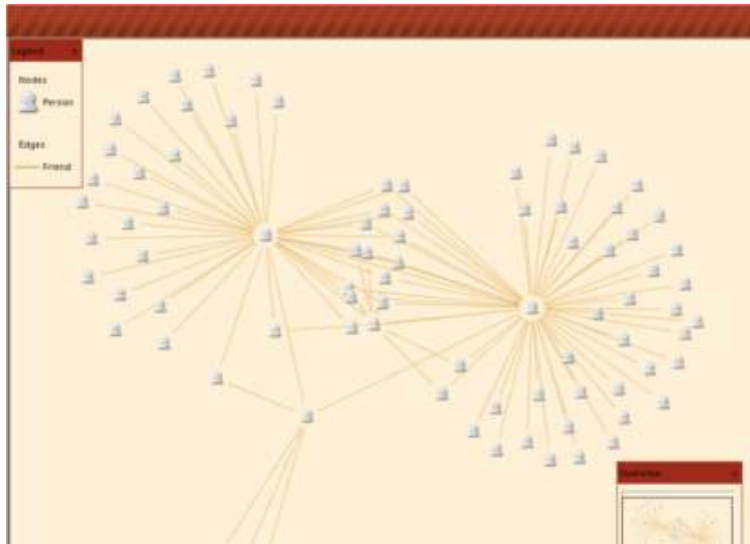


Figure 3. Extended graph.

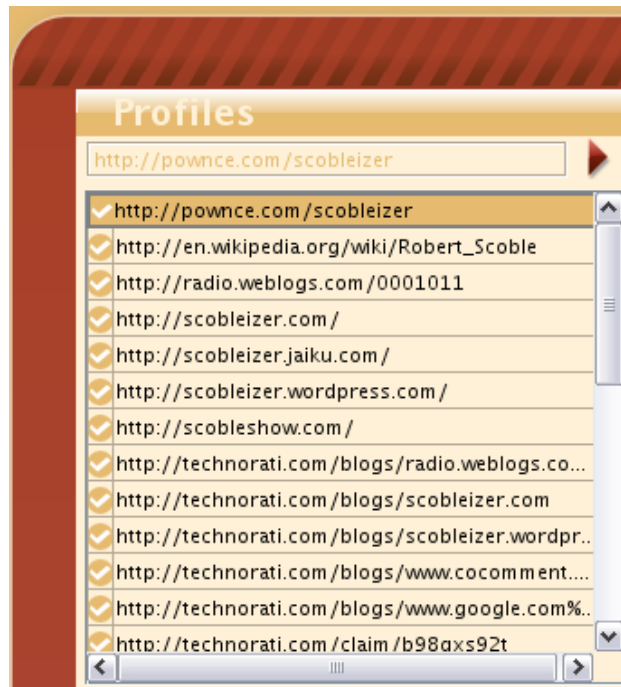


Figure 4. Profiles found that match a user's given URL.



Figure 5. Searching for other profiles.

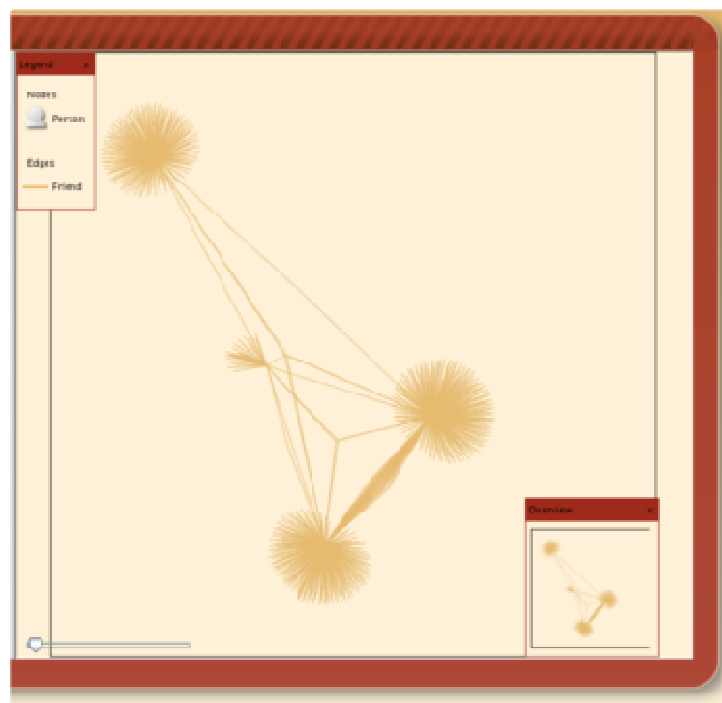


Figure 6. Network of users with large amounts of friends. Note this network took some time to build.

References :

- 1 <http://code.google.com/apis/socialgraph/>
- 2 <http://gmpg.org/xfn/>
- 3 <http://www.foaf-project.org/>
- 4 <http://www.altio.com/>
- 5 <http://json-lib.sourceforge.net/>