Date: 11/07/2012

Challenges Faced Developing PRISONER's First Study

Introduction

As is to be expected with any software project, a number of challenges were encountered whilst developing PRISONER's Facebook gateway and the first web study using the framework.
This document aims to outline any such difficulties and detail any solutions or workarounds which were implemented.

Facebook Graph API – Performance

One problem that was encountered early on in the project relates to Facebook's Graph API. Although the API itself is straightforward and fairly intuitive in most cases, it can be extremely slow at times, often for no apparent reason. Retrieving data such as status updates, photo albums and photos was consistently sluggish, often taking in excess of five minutes to get the aforementioned pieces of data for a single user.
To avoid exposing users of PRISONER to this slowness, Luke added caching and asynchronous calls to the framework. This allows applications to send out requests for all the data they need and then periodically poll PRISONER. If the data is yet to arrive PRISONER will return an empty response, otherwise the data that was requested is sent in the response as a JSON-encoded object.
This workaround effectively gets around the performance issue and should benefit other social network gateways in the future.

Facebook Graph API – Inconsistencies

One of the requirements of the Facebook gateway was that it could retrieve a user's status updates. As we make no assumption about what sort of studies will be undertaken using the PRISONER service, getting as much information as possible about status updates is clearly desirable. One such piece of information that could be useful in a great many studies (Including ours) is the **privacy** attribute. This is where an inconsistency on Facebook's part was uncovered.
Typically, to get a user's status updates, a call to the `/statuses` connection would be made. This worked well, and we were able to get each and every one of a user's status updates this way. However, privacy information was not available using this method.
After some research, it was found that statuses were also available by making calls to the `/feed` connection. This connection returns everything that has been posted to a user's wall, including their own status updates, and also includes privacy settings for certain objects. Clearly, therefore, this would be ideal for retrieving status updates along with their associated privacy attribute.
However, after a few tests, both Luke and I noticed that status updates were only returned from mid January 2011. Other items, such as wall posts by friends, were all returned.
This was submitted as a bug to Facebook and, as of **11/07/2012** has a **low** priority.
It was decided to continue to use the `/feed` connection as we felt it was more useful to have fewer data complete with privacy information than it was to have lots of data with no privacy information.

Facebook Graph API – Check-ins

For each piece of data we retrieved from a user's Facebook account, we ideally wanted a permalink to that piece of information so that the user could see it on Facebook. This was possible with data such as status updates, photos, friends and even profile information. However, it was not possible for check-ins.

Our preferred way of obtaining check-ins makes a call to the `/locations` connection. This connection is called **objects with location** and is a more accurate indicator of where a person has been as it includes cases where a user has been tagged in a photo and that photo has a location tag, instead of just including explicit check-ins made by the user. However, using this connection it is not possible to generate a permalink for the check-in without effectively recursing through object IDs. This would have resulted in a dramatic hit to performance, as well as having to deal with all the different types of check-in object and how they could be given a permalink. (Eg: Photos, videos, explicit check-ins, check-ins by friends)

Web Questionnaire – Assigning Groups

This seems like a straightforward task – just give people groups based on their Facebook ID or the last person to complete the study – but it actually proved to be quite difficult and still is not perfect.

One of the problems relates to the fact that the **very first** piece of information seen by a prospective participant depends on their group. This means that groups have to be assigned before we know whether or not a participant will actually take part in the study.

Another problem is that participants may get part of the way through the study and then decide to stop. Combining these two problems means that, in the worst case, every person who successfully completes the study could potentially be assigned the same group.

In order to try and avoid this, a best-effort algorithm was used. This essentially assigns groups by checking to see how the last participant was grouped, and then assigning the other group. In addition to this, once a participant successfully completes the study, a flag is set indicating that the next participant should be assigned the other group. (Again, it is possible that subsequent participants may choose not to take part)

In the field, this approach seems to be working fairly well, with a ratio of **17:16** as of **11/07/2012** just after 2pm.

Web Questionnaire – Periodic Apache Crashes

Periodically, it would appear that some component in PRISONER encounters a segmentation fault which causes Apache to restart. This deletes all session keys from memory and, in the worst case, would mean that participants are unable to complete the study as the web app cannot retrieve their Facebook data. (The key used to request data will be viewed as invalid by the server and no data will be returned)

A workaround for this was integrated into the web application which is effectively a "limp mode." If we detect that a participant has been on the "Loading…" screen for at least one and a half minutes, the web app then attempts to use whatever information we have available in order to generate questions. For example, if Apache crashed before status updates could be retrieved, extra questions would be generated from profile information, photos, check-ins, likes and so on. Extra questions are distributed evenly amongst whatever data is available to avoid asking too many questions about the same type of data.

Web Questionnaire – Evenly Distributing Questions

Without the asynchronous calls to PRISONER, evenly distributing questions over the different types of data was trivial. However, the asynhcronicity introduced a great deal of complexity. This is because the web app needs to assign a preferred number of questions of each type without knowing how much data is actually available. To solve this problem, the application assigns questions in intervals. All calls for data are sent at once and, before the participant starts the study, a number of basic questions are generated to keep them busy while more data / questions are arriving in the background. Each time the participant clicks to move to the next question, the web app checks to see whether or not any new data has arrived. If it has, new questions are generated as necessary. This process repeats until all the different data types (Status updates, photos, etc.) are returned.

If some types of information are lacking, compensation is acquired by evenly distributing questions amongst the remaining data types as mentioned in the previous section.

In some cases, the participant can complete each of the available questions before all their data becomes available. In this case, they are presented with a loading screen to assure them that the web app is still working and to give the web app time to receive the data.