SocWeb - Search Within Your Social Networks
Technical Report
Supported by Marie Curie Grant
PIRG06-GA-2009-256603

Fotis Psallidas, Alexandros Ntoulas, Alex Delis
{fpsallidas, antoulas, ad}@di.uoa.gr

October 2012

Contents

Chapter 1

Introduction

The share of social activity by individuals has become a new world trend. The amount of data produced at any second has exceeded the limit of information that any human mind can keep track off. Even nowadays, that the computation power has been taken to its limits and we are able to safely keep track of this information in large computing and storing systems, the task of acquiring knowledge from this huge amount of data has become a very tough and intensive task. Moreover, all this data produced both in structured and unstructured ways has formed a knowledge database that each individual can conceive in a different way. Each type of information published at any point of time can be perceived differently among different users. Just like in ordinary life where people with different perspectives, background, ethics etc filter the received information based on their criteria, exactly the same applies to social networks.

The contribution of our work is to describe the several problems that emerge in the field of search engine design and implementation using diversified content from social networks. The final outcome of our work is a user-driven social network search engine, namely SocWeb, that allows users to search within their social networks and provides personalized results that users seek for based on the provided search terms.

The problem of search engine design and implementation has been active for several years now and thus has been well studied by the community. However, social networks contribute a new dimension in the field that search engines should deal with to provide the information needed by individuals. Those networks and the information that lies within them add multiple constraints that contemporary search engine strive to deal with former techniques instead of attempting to define and deal with the new problems that emerge. Thus, they fail to provide up-to-date knowledge needed by individuals. These problems are the type of information, the extraction ways in respect of the several protection levels on top of that information, the scale of the data produced by both individuals and machines and the diversity of social networks purpose and structure . In our work we address all these problems and suggest generic ways to overcome them.

In chapter 2, we introduce the core concepts of the SocWeb project, the *SocWeb Model* which is a generic way to describe arbitrary social network graphs, ,the *SWODL* (abbreviated for SocWeb Object Definition Language) which is a language that we introduce to describe object types of social network graphs and the *SocWeb Generic API* which makes use of the SocWeb Model and SWODL templates to provide a very simple interface to retrieve information from these social networks.

In chapter 3, we present the architecture of each component of SocWeb. First, the Application Level that provides a simple but rich front-end to the user and triggers specific requests to be handled by the backend. Second, the Distributed Crawler level, which is cosnidered the most impartant part of SocWeb, and is able to retrieve information, using the core concepts introduced in chapter 2 and specific policies that allows it to decide what and why to retrieve next efficiently. Third, the Storage System, which takes the information extracted from the Crawler, apply several checks and finally store it in a distributed manner. Fourth, the Indexing level which reads the output of either the Crawler directly or from the Storage System and creates the final indices that allow efficently query performing by the users. A really important approach that emerges from our design is that each componenet is considered stand-alone. Its policies dont affect the policies of the other components, because the communication among them is formed based on the core concepts that each component is able to understand and act upon them.

Chapter 2

Core Concepts

2.1 Overview

2.2 SocWeb Model

Each Social Network can be seen as a bidirectional graph containing objects of diversified types and links between these objects. Here, we define the terminology that we use to describe them. SocWeb Model defines 2 generic types of objects:

1. ***primitive* objects**, containing only in-links,
2. ***composite* objects**, containing both in- and out-links

Where an object A has an in-link from object B or equivallently object B has an out-link to object A , if object A can be reached from object B .

Each object has a ceratin type, named ***SocWebObjectType*** that corresponds to the type of the object in the social network graph. These socWebObjectTypes are predefined in *SocWeb* and denote the type of objects supported. The form of each object type ,that we make use of, is in compressed mode in order to denote both the type of the object and the social network that it corresponds. For instance, a Facebook photo's socWebObjectType is FB_PHOTO and Twitter photo's socWebObjectType is TW_PHOTO. For primitive objects, we currently support SW_STRING and SW_INTEGER that stand for string and integer fields accordingly.A full list of all socWebObjectTypes currently supported can be found in [TODO: APPENDIX A]

The primitive objects are the indispensable pillar of the social network graph, because in essence they define its boundaries. On the other hand, composite objects have links to either primitive types, or composite types or even collection of composite types.

To decribe this diversity of links, we define 3 different types of them:

1. links to primitive objects, named ***p-links***,
2. links to composite objects, named ***c-links***,
3. links to collections of either primitive or composite objects of the same type, named ***co-links***

We also say that a composite object C1 is:

1. ***p-connected*** to a primitive object P, if there is p-link from C1 to P,
2. ***c-links*** to a composite object C2, if there is a co-link from C1 to C2,
3. ***co-links*** to a collection of composite or primitve objects of the same type CO, if there is a link from C1 to CO.

For a composite object, its p-connected primitive objects constitute its attributes while the c-connected composite objects,that is linked to, resemble its relationships. The co-connections, in essence, are c-connections of the composite object with multiple objects of the same type thus describing multiple relationships of the same type.

In order to define programatically the various objects we make use of inheritance. We introduce the generic socWebObject class which is illustrated below in Listing 2.1

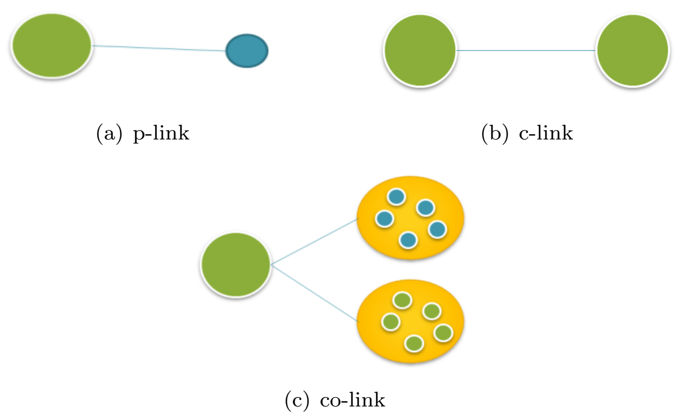

Figure 2.1: Different link types supported by SocWeb Model. Composite objects are denoted by green circles where primitive objects by blue ones.

Listing 2.1: socWebObject class

```
class socWebObject{
public:
  typedef __gnu_cxx::hash_map<std::string, int, char_HashFunc,
      char_eqstr> TMAP;
  typedef __gnu_cxx::hash_map<std::string, json_spirit::mValue,
      char_HashFunc, char_eqstr> VMAP;

  TMAP* tmap;   // the description of the object
  VMAP* vmap;   // the objects that corresponds to the above
      description


  std::string socWebId; // the id of the object in socWeb
  socWebObjectType objT; // the type of the object

  socWebObject(std::string _socWebId);
  socWebObject(std::string , socWebObjectType);
  socWebObject();
  socWebObject(socWebObjectType _objT);
  socWebObject(const socWebObject&);
  ~socWebObject();
  virtual socWebObjectType type();
  virtual void print(int level=0);
  virtual std::string get_socWebId();
};
```

Using this generic class we are able to create derived classes either for primitive or composite objects.For instance Listing 2.2 presents the socWebString class that refers to SW_STRING primitive type and Listing 2.3 present socWebFbPhoto that refers to FB_PHOTO type.

Listing 2.2: socWebString class

```
class socWebString:public socWebObject{
  friend std::ostream& operator<<(std::ostream& output, const
      socWebString& p);
public:
  std::string str;
  socWebString(const std::string&);
  socWebString(const int&);
  socWebString();
  void operator =(const std::string& _str);
  void operator =(const int& _num);
private:
};
```

Listing 2.3: socWebFbPhoto class

```
class socWebFbPhoto : public socWebObject{
public:

  //P-links
  socWebString id;
  socWebString name;
  socWebString icon;
  socWebString picture;
  socWebString source;
  socWebString height;
  socWebString width;
  socWebString link;
  socWebString  created_time;
  socWebString  updated_time;
  socWebString  position;
  socWebInteger comments_cnt;
  socWebInteger likes_cnt;

  //C-links
  socWebFbFrom* from;
  socWebFbPlace* place;

  //Co-links
  std::vector<socWebFbPhotoTag*> tags;
  std::vector<socWebFbLike*> likes;
  std::vector<socWebFbComment*> comments;
  std::vector<socWebFbPhotoImage*> images;

  socWebFbPhoto(JsonHandler);
  socWebFbPhoto(const socWebFbPhoto&);
  ~socWebFbPhoto();
  void print(int level=0);
};
```

The main objective goal of SocWeb, apart from describing inter-connections within a social network, is to be able to describe ”arbitrary” social networks and provide the ability to define connections among different objects from different social networks. Using this abstract model we are able to address the question of how to describe, efficiently and in an easily adaptable manner, “arbitrary” social networks. For each social network we provide SocWeb with up-to-date definition of all of its object types along with their p-, c- and co-connections. SocWeb is able to read these descriptions and create subclasses of the socWebObject generic type. The definition of the object types can be generated with a special, fairly simple language that we introduce in section 2.3, the ***SWODL*** (abbreviated for SocWeb Object Definition Language).

Finally, we deal with the problem of connecting objects of different networks by introducing a new link type, named s-link. S-link connects objects of different networks, thus making our arbitrary network graphs connected into a single graph. Consequently we have a way to extract aggregated information for an object, that lies in one social network, using its s-connected objects from another social network. The concept of s-link along with the other type of links is illustrated in

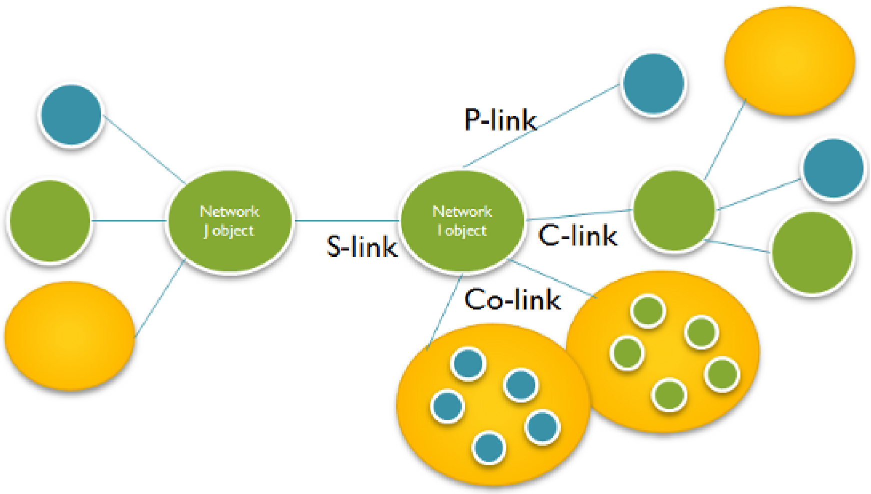

Figure 2.2: S-link concept along with the other type of links defined in SocWeb Model

2.3 Generic SocWebAPI

Social networks' Web pages ,that contain the information needed, are a simple picture of the information contained in these social graphs. Thus, instead of striving to acquire information from these pages in order to reproduce parts of the social graphs we take a different approach. Nowadays, all the large social networks provide APIs (abbreviated for Application Programming Interface) that give access to their internal graph. The first step, in order to use the API provided by a specific social network is to register an application in this network. Then, the users of this registered application authorize the app with certain authorization privilleges that the application requests. Next, the application ,based on the authrorization that it is granted, is able to use the corresponding API in order to retrieve the information needed. The whole process is illustrated in Figure 2.3

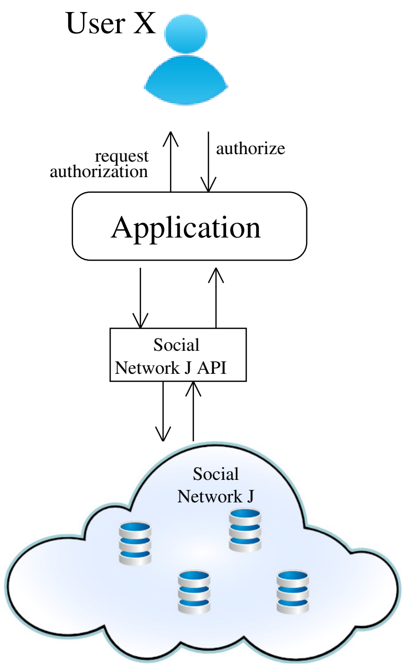

Figure 2.3: Application authorization and Information Retrieval from Social Network J using the corresponding API

Although the above schema can be used to perform per authorized user crawling it can't scale in a generic way. The amount of social networks has increased tremendously, each with its own meaning, purpose, API functions, results structure-type and corresponding parsing techniques, authorization procedures, per ip-per user and per application call limitations. First, in terms of meaning and purpose, the information that lies within each social network reveals the meaning and the purpose of the social network itself. The usage of their APIs reveals the information need applied to that specific meaning and purpose. For instance, the meaning of a facebook photo album largely differs from a flickr photo album because the meaning and purpose and consequently the usage of facebook and flickr differ. Thus, if a developper needs to retrieve photo albums in general should be aware of the meaning and purpose of the social network that she retrieves from. Second, the API functions differ among social networks. Developers that thinking of dealing with multiple social network APIs are about to face myriad API functions where each function requires multiple parametrization. Third, the type of the results differ among social networks. For instance, some social networks rely solely on JSON formats, while others rely only on XML format, thus making parsing really hard. Fourth, each social network has its own authorization procedure. Most of them rely upon different versions of OAuth protocol. Based on the OAuth protocol there are also 3 different kinds of authorization procedure each corresponding to either web, desktop or mobile applications. Of course, social networks nowadays also provide API calls that need no authorization.Thus there is an extra level of complexity to determine whether or not an API call needs authorization or not. Morover, if all the API calls that are about to be used by the application need no authorization the developper doesn't need to implement the authorization step. But, this needs a priori knowledge of the final calls that are about to be used. Finally, each social network poses (or not) its own limitations on the amount of requests that can be handled by their severs. The limitations are applied per ip, per hour, per authorized user and per application and vary among different social network (They even differ within the social network itself, e.g. Twitter has different limitation on the usage of REST API and different limitations by the Streaming API). The industry of application in the field of social networks are affected by all the above problems. Developpers have to deal with multiple hurdles which are not the main focus of the final commodities.

What developpers really want afterall is to retrieve certain objects from the social graphs. We adhere to this challenge and based on our SocWeb Model and the SWODL we propose a generic API, Generic SocWebAPI, with only 4 family of API calls that is able to retrieve any kind of information from any social graph available out there. We call this Generic SocWebAPI. The API function calls are:

1. `socWebObject get_object(object_id, object_type)`

   Retrieve the object with id object_id and type object_type. All its p-links, c-links and co-links will be retrieved for this object.

2. P-link family call
   Retrieve only the p-connected objects of an object.

   - `socWebObject[] get_plinks(object_id, object_type)`
   - `socWebObject[] get_plinks(object_id, object_type,  attribute[])`
   - `socWebObject[] get_plinks(object_id, object_type,  time_range[])`
   - `socWebObject[] get_plinks(object_id, object_type,  attribute[], time_range[])`

3. C-link family calls
   Retrieve only the c-connected objects of an object.

   - `socWebObject[] get_clinks(object_id, object_type)`
   - `socWebObject[] get_clinks(object_id, object_type,  attribute[])`
   - `socWebObject[] get_clinks(object_id, object_type,  time_range[])`
   - `socWebObject[] get_clinks(object_id, object_type,  attribute[], time_range[])`

4. Co-link family calls
   Retrieve only the co-connected objects of an object.

- `socWebObject[] get_colinks(object_id, object_type)`
- `socWebObject[] get_colinks(object_id, object_type,  attribute[])`
- `socWebObject[] get_colinks(object_id, object_type,  time_range[])`
- `socWebObject[] get_colinks(object_id, object_type,  attribute[], time_range[])`

The general form of the API calls is

Listing 2.4: API's functions' general form

```
    socWebObject[] get_<Type Of Link>(object_id , object_type , [
        attribute[] ,time_range[]])
```

For each call an application should provide the object_id of the object to be retrieved and its type. The type of the object should be collected from a list of object types that we provide and is illustrated in [APPENDIX A]. Moreover, an application might want some of the p-links, c-links, co-links of the objects returned. This can be achieved using the name array parameter, allowing the caller to implicitly declare the p-links, c-links, co-links needed. In addition, most social networks APIs retuned results based on a specific time range.For instance, Facebook provides the ability to return user's posts within a specific time range. If a time range request is not available for the specific connection of the object then the most recent will be returned. Finally, callers of the Generic SocWebAPI to request results for specific p-links , c-links, co-links in conjuction with time ranges.

As it can be seen all these API calls return arrays of socWebObjects. Sometimes, the results that can be retrieved have really large size thus making these synchronous calls quite slow. We adhere to this problem because it arises several technical challeneges and present a cursor-based solution. All the API calls listed above accept another parameter at the end of the parameter list that indicates the amount of results to be returned.Consequently the new general form of our API calls is

Listing 2.5: API's functions' general form with limit

```
socWebObject[] get_<Type Of Link>(object_id , object_type , [
    attribute[] ,time_range[]] , limit=-1)
```

The *l*imit=-1 above indicates that all the results should be returned. A positive limit means that the amount of results must be limited by the value of the limit. In order to retrieve results after the above limit a new API call is introduced

Listing 2.6: Get Next Results

```
socWebObject[] get_next(limit=-1)
```

where the limit again refers to the amount of results that get returned.

Finally, by using our new Generic SocWebAPI the information retrieval from "arbitrary" social networks is illustrated in 2.4.

To the best of our knowledge, there is no previous work that combines all the APIs of the most popular famous. Developpers have the ability not to concern about the various APIs and the way should be handled. They just use a simple API with 4 families of function calls that gets the work done efficiently. The

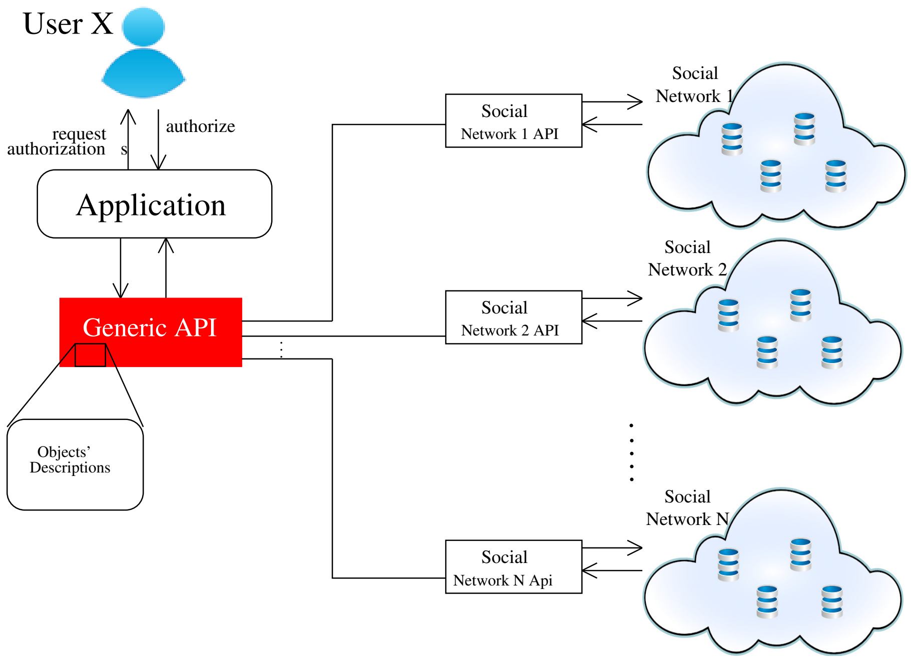

Figure 2.4: Application authorization and Information Retrieval from Social Network J using the corresponding API

only concern is the description of the objects needed to be retrieved, thus we keep an up-to-date repository of all the possible descriptions to use them at any time.

Chapter 3

Architecture

3.1 Application Level

3.2 Distributed Generic Crawler

3.3 Storage System

3.4 Indexing

Chapter 4

Experiments

Chapter 5

Conclusions

Chapter 6

Future Work

Chapter 7

Acknowledgments