Viewing FOAF – Development of a Metadata Explorer

Josef Petrák

Faculty of Informatics and Statistics, University of Economics, Prague
Winston Churchill Sq. 4, 130 67, Praha 3, Czech Republic
jspetrak@gmail.com

Abstract. Social networks are widely accepted application of Semantic Web technologies and are also interesting for general public. Nowadays there is a lack of quality user-friendly browser which could express meaning of the metadata stored in the most of FOAF profiles and present this knowledge in human-understandable form. The aim of the article is to inform about development of the AFE (Advanced FOAF Explorer) which is intended to perform these services and supersede one similar project.

Keywords: social networks, Semantic Web, RDF, FOAF, PHP5

1 Introduction

Friend Of A Friend vocabulary (FOAF) [1], an ontology used for description of personal profiles and relations among people, is well-known and popular in the Semantic Web [2] community. Thank to the user-friendly tools such as FOAF-a-Matic [5] people who are not familiar with RDF [14] can create their own profiles and publish them on the Internet. This is a good because may start spreading of the Semantic Web technologies to the public.

But if these users create such profiles, they also require having a chance to view their profiles or to browse the profiles of other people. Nowadays there are several "viewers" but their features are limited. These restrictions do not allow to wide spreading of FOAF users. These limitations are discussed in section 2.

The aim of the project "Advanced FOAF Explorer" [9] is to develop an FOAF explorer with user-friendly XHTML [17] output. To allow the developers to easily extend the code and add support for new vocabularies which may extend existing ontology. The most important task is the show relations among various resources because this is the main positive of using FOAF.

I already developed a first beta version based on PHP. It shows randomly chosen subset of FOAF terms and relations defined in the ontology extension called "relationship" [13]. The structure of the output shows how should be the data presented. History of this version and details about implementation are discussed in the section 3. Further development should produce new version of this application. The basic ideas and implementation details are described in the section 4.

V. Snášel, K. Richta, J. Pokorný (Eds.): Dateso 2006, pp. 74–84, ISBN 80-248-1025-5.

1.1 Brief introduction into social networks

If we consider definition of social networks published on the Wikipedia [3], we have to restrict the term to on-line social networks. It is an on-line community based on a Website which allows each member to communicate, make friends among other members, to discuss topics which are interesting to this community … FOAF allows to store information about member of any community, the relationships among them (and type of the relation such as friend of someone, parent of someone, enemy of someone, roommates with someone …). This is interesting for most of the people because making contacts, making friends and entertainment are the basic needs of almost everyone.

1.2 What is FOAF?

The basic explanation is that FOAF is ontology. It is defined using RDF Schema [18] and OWL [15]. If you see the specification [13], you can find there a lot of classes and their properties. Using them you can create various RDF[1] statements about you, your relatives, friend and the people who you know. You can describe your work, schools, interests, your Websites and if you use some additional modules, you can also describe countries you have already visited, languages which you reads, speaks or writes and many other more or less interesting information.

Let's have a look at the basic structure of a FOAF profile. In the Figure 1. I gave an example in XML [16] syntax of RDF. It contains some information about a person, and statement that this person knows somebody else.

As you can see, the FOAF defines terms for commonly required properties such as name, mailbox, and gender. But the specification defines many more of them such as chat IDs for instant messaging services (AIM, ICQ, Jabber, Yahoo!, and MSN), it allows to state, that any person has a homepage, a weblog, to link depiction, personal interests. The most interesting property is `<foaf:knows>` which states that a person knows any other person. The type of the relation is not defined. If you want to explicitly define the quality of the relationship between to persons, you have to use extension RELATIONSHIP [13] which contains a lot of interesting sub-properties, e.g. previously mentioned friend of, enemy, of, roommates with and many other such as works with, employees, knows by reputations, sibling of, loves, … The complete view about the vocabulary you can make reviewing the specification.

1.3 How to display it on the Website

There are some technical limitations which do not allow to directly put the FOAF profile into the code of any Website. You can store your profile in the file and link it to the page using element `<link>` with parameter `rel="meta"`.

[1] More information about *Resource Description Framework* and also the specification you can find at http://w3.org/RDF

```
<?xml version="1.0" encoding="UTF-8"?>
<rdf:RDF
  xmlns:foaf="http://xmlns.com/foaf/0.1/"
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#" >
  <foaf:Person rdf:ID="adam-smith">
    <foaf:name>Adam Smith</foaf:name>
    <foaf:gender>male</foaf:gender>
    <foaf:mbox rdf:resource="mailto:adam@smith.name" />

    <foaf:knows>
      <foaf:Person>
        <foaf:name>John Doe</foaf:name>
        <foaf:knows rdf:resource="#adam-smith"/>
      </foaf:Person>
    </foaf:knows>
  </foaf:Person>
</rdf:RDF>
```

Fig. 1. The example of basic structure of FOAF profiles.

2 Motivation for development

To have a complete view to the history we have to explain why the development of this explorer started was, and why it is called to be “advanced”. In the years in which the FOAF itself was created there was a need for quality and extendible browser which could serve information stored in such profiles to the user in XHTML format. There are several applications used for parsing in viewing these profiles, e.g. FOAF Web View [7] or Plink.org [8] which also integrated a simple storage for viewed metadata. But they have various problems. First of all they do not interpret all or at least most of the FOAF terms. If an application interprets all terms, its output does not have a nice look or the information are presented in illogical order so the user is confused and cannot find what he is looking for. They often do not show relations among various resources (people, people and projects ...). Another problem is that FOAF allows easy extendibility and there are many widely used 3rd party extensions. Usually these browsers do not support these extensions. And the last problems is that the interface does not explicitly explain the meaning of the terms and do not allow internationalization so the non-English speak users cannot use them.

Nowadays the most used application is FOAF Explorer [6] developed by Morten Frederiksen. It is based on XSL Transformations. It supports various extensions, displays some relations among the resources. But the look of the output could be created better, show more information in a user-friendlier way. And this is chance for the project of AFE!

3 History of the project

The development was started by two students – Josef Petrák and Michal Trna - in the summer 2004. The very first goal was to quickly create first beta-version for viewing of authors' profiles. Further development should have focused on cleaning the application code and to implements most commonly used FOAF extensions. Unhappily its development was frozen in the same year. This beta version is still available [9] and the community was informed about it on the IRC channel [4].

3.1 The first beta – details of the implementation

Parsing of the RDF files is done using the library RDF API for PHP [11] which is broadly accepted library for manipulation with RDF files in the PHP applications. The scenario of producing result is easy.

1. User writes URL of any profile which he wants to browse into the form.
2. Application tries to download the FOAF file. If it is successful, it parses the RDF and generates in memory a RDF graph.
3. To simplify later generating the output, the application transforms a special structure from the graph. In fact the structure is an 3D associative array where the indexes represents:

   – URIref of the resource
   – URIref of the property describing the resource
   – Auto incremented integer identifying the values of the property.

   The algorithm of creating the array structure is shown in the Figure 2.

After that, huge function `format_person($data, $model, $id_person)` goes through this array and for each resource generates XHTML code which will appear in the output page. It also generates anchors which allow the user to click on it and see the relations between the resources (on the figure 3 there is shown simple output from an existing profile). You can imagine that this solution is quite "dirty", it lacks any design patterns but it was chosen for quick implementation of the problem. And as we can see, it works well.

```
function get_data_structure ($model) {
  $res = array();

  for ($iter = $model->getStatementIterator();
                              $iter->hasNext();) {
    $statement = $iter->next();
    $predicate = $statement->getLabelPredicate();
    if (!is_array($res)) {
      if (!is_array($res[$predicate])) {
        $res[$predicate] = array();
      }
    }
    $res[$predicate][] = $statement->getLabelObject();
  }
  return $res;
}
```

Fig. 2. The algoritm for generating array structure from a RDF graph.

Now is time to change the implementation, clean-up the design and remake the output. Future plans are all described in the section 4.

4 Future plans

4.1 MVC Design Pattern

In any well-designed application it is necessary to separate data model, view layer and logic which loads and manipulates with the data. In modern programming languages such as Java or .NET there various frameworks, which allows simple use of the pattern of **Model – View – Controller**. We can mention *Struts*[2] and *JavaServer Faces* in Java, package *Web.Forms* in .NET. For PHP there are only limited amount of such frameworks. The most advanced is component-driven framework called PRADO[3], but it is too complex to be easily implemented. Better way is to implement own simple MVC solution. The new library is based on PHP5 object model. All important components are defined using interfaces andd errors are handled using exceptions.

[2] http://jakarta.apache.org/struts/
[3] http://www.xics.com/

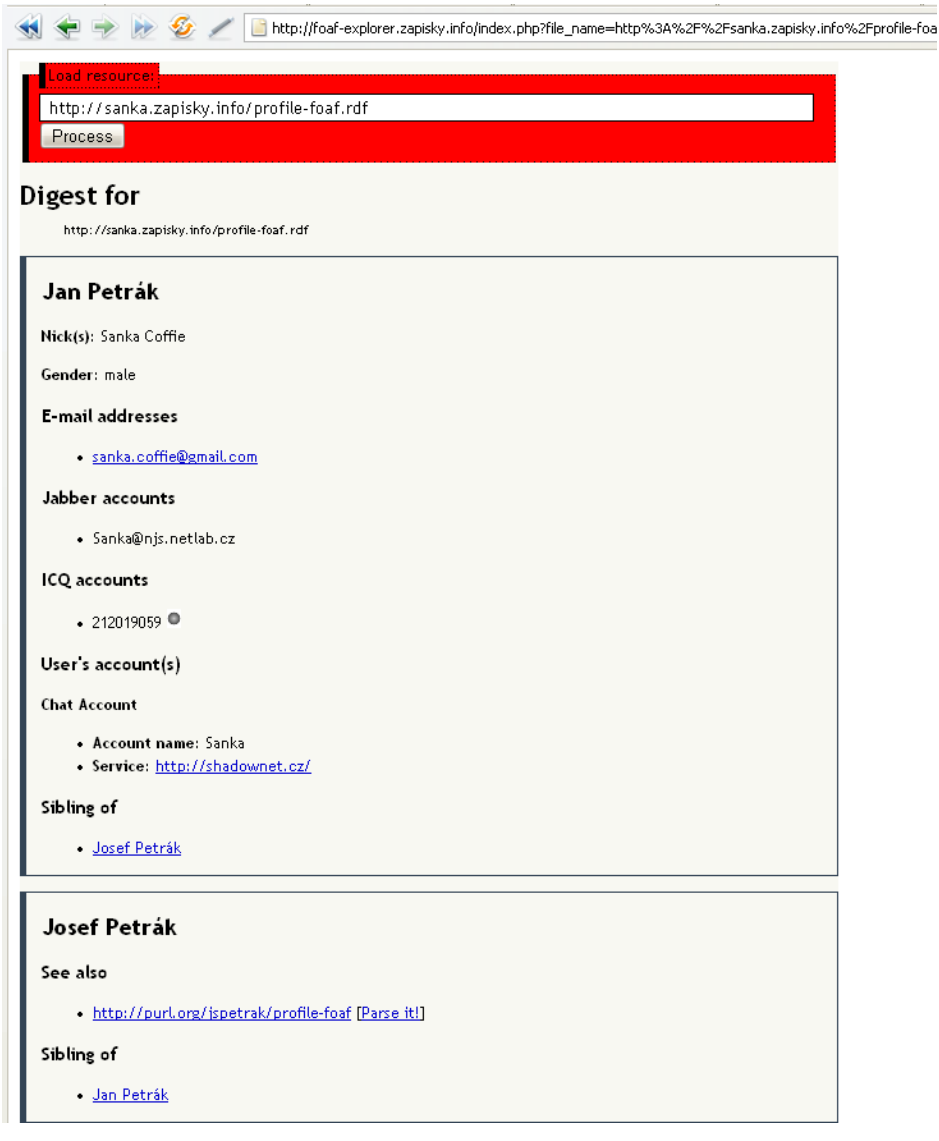

Fig. 3. An example of output from the first version of Advanced FOAF Explorer. The address of the profile is shown in the address bar. Clicking on the link "Parse it!" we can show other linked profiles in the viewer.

Each application is started using static method, such as `AfeApplication::run();`, the application will recognize name of template from the PHP file name. For `index.php` is assigned the template

`views/_index.php`. The templates are written also in PHP using `foreach` construction and using getter methods of all objects from data model.

The application class manages all operations. It starts loading resources; stores got data in any model object and the calls a viewer which uses given template and according to the instruction views the information. The core class also catches any thrown exception and accesses it by a getter method. Using getter and setter methods is the basic concept of the design of this class – it makes the API clearer.

There is a class which parses all information from server configuration variables, request variables (information sent by GET or POST methods) as well. Further development of this MVC framework will introduce common configuration files in INI or XML[4] format.

4.2 History of viewed profiles

To control viewed files and have a chance, how to explore unknown extensions of FOAF, it is necessary to store the data about browsed files. To maximally simplify the task, it is designed simple database table containing three columns: URI of the viewed file, date and time when the request was received from a user and count of triples stored in the profile. Data may be store in any relation database but the best for simple use in PHP5 are integrated database SQLite or MySQL 4.1 Both of them offers object-oriented approach to the API (through the `SQLiteDatabase`, or `mysqli` object respectively). The structure of the database table is shown in figure 3.

```
CREATE TABLE ‘history’ (
  ‘uri’ VARCHAR(255) NOT NULL,
  ‘dt’ DATETIME NOT NULL,
  ‘triples’ INT UNSIGNED NOT NULL,
  PRIMARY KEY ( ‘uri’ , ‘dt’ )
);
```

Fig. 4. Structure of SQL table for storing the history

To create fully persistent approach, there is a class `History` in the application with methods for loading and saving the data. If it is necessary to change the database, the implementation will be changed only in this class and the rest of the application will be not influenced by this change. The question is if implement this class straight or define an interface and implement classes using drivers for all considered database drivers (which were mentioned previously).

[4] eXtensible Markup Language

To complete the persistence, there is the object `HistoryItem` which represents one row from the table. It defines getter and setter methods for manipulation with the data in the object.

4.3 Data model

The biggest change from the first version is the data model. Due to ineffective work with the solution of an associative array it was left and now fully object – oriented solution is going to be created. The basic concept is to represent all known RDF resources as classes and their properties as getter methods which return array with values of property of the same type. We can also describe the hierarchy of the resource, e.g. that `<foaf:Person>` is a subclass of `<foaf:Agent>` (using keyword `extends`). It means, that Person may have all of properties, which are defined for the Agent and it also may have some new properties which the Agent does not have. It also allows easily extend the application model. Changing the properties or creating new classes is a question of seconds.

In the Java, the class in the hierarchy of classes is `java.lang.Object`. Also our data model has a super class which defines common methods and properties. They are common for all types of resources. This class represents the resource `<rdf:Resource>`. There is also interface `Labelable` which defines one common methods called `getCommonLabel()`. This method offers label which will appear in the name resource (such as headers or image titles). The code of this interface and the super class `RdfResource` is shown in the figure 5.

As you can see, the method `__toString()` is used for testing purposes. It returns the dump of structure of the class. The most interesting method is `parseRequiredProperty()` it has to be called in the constructor of the model classes for every property which we intend to read from the RDF graph. Note down that it is necessary that if we declare any subclass, we have to send the parameters to the super class otherwise it will not be loaded values of properties defined in the parent classes.

```
interface Labelable {
  public function getCommonLabel();
}
class RdfResource implements Labelable {
  private $uriRef;
  private $rdfType;
  private $foafName;
  public function __construct(&$model, &$resource) {
    $this->uriRef = $resource;
    $this->parseRequiredProperty($model, $this->rdfType,
RDF::TYPE());
    $this->parseRequiredProperty($model, $this->foafName,
FOAF::NAME());
  }
  protected function parseRequiredProperty(&$model,
&$objectProperty, &$property) {
    for ($i = $model->find($this->uriRef, $property, NULL)->
getStatementIterator(); $i->hasNext();) {
      $stmt = $i->next();
      if (is_null($objectProperty)) $objectProperty =
array();
      $objectProperty[] = $stmt->getLabelObject();
    }
  }
  public function getRdfType() { return $this->rdfType; }
  public function getFoafName() { return $this->foafName; }
  public function __toString() { return print_r($this, true);
}
  public function getCommonLabel() {
    if (count($this->foafName)>0) return explode(', ', $this
->foafName);
    else return $this->uriRef->getUri();
  }
}
```

Fig. 5. The code of the `RdfResource` class and the interface `Labelable`

4.4 Internationalization

Internationalization allows non-English speaking users to use this application. We have to separate labels from the index and output page, store them in one place and create multiple translations. If we consider existing solutions for internationalizations, there is mechanism represented by function `gettext`[5]. It is standard for PHP application so it is not necessary to create any other now framework.

There is also one important problem. We have to find volunteers who can translate these texts into languages different from Czech or English. In sure, that if we announce new version on the FOAF IRC [4] channel and inform about possibility to translate the user-interface, we will find the volunteers who can help us to translate the UI into their mother language.

4.5 Data-binding

We considered implementing a mechanics of data-binding which could automatically generate data model classes from given RDF Schemas and ontology. Even if it is a good idea, we did not find any existing tool for this purpose. Our project is not targeted to create any kind of this application. But it could be useful for developers from the Semantic Web community. There is space for other developers to implement it …

One problem of this solution could be that the definition of FOAF ontology and its modules are stored in different files. Some of them are Ontologies in OWL and some of them only RDF Schemas. So this tool should generalize the generate model and ignore some constructions which are not common for RDFS and OWL (and its versions).

5 Conclusion

It is necessary to define all necessary objects in the data model and to consider which extension support – the most important extension is the module RELATIONSHIP, which extends the basic concept of making relations among people defining a lot of new sub-properties. After that there should not be any problem which could slow down the development. The most important goal of this project is to show that building of any Semantic Web application is not so hard as many of developers think.

After releasing the tool and announcing it to the international FOAF community we expect big interest in the source codes, details about the implementations and volunteers who can create multiple translations. The aim of the project is to pay attention to the Semantic Web, especially FOAF and to promote its using among ordinary (understand non-programmers) users. After finishing all of these tasks, we

[5] Documentation of gettext: http://php.net/gettext

can focus on improving our MVC framework, to the development of other RDF tools which may help our author easier build their applications based on RDF graphs.

References

1. Project *Friend of a Friend*: http://foaf-project.org/
2. Wikipedia. Term *Social Networks*: http://en.wikipedia.org/wiki/Social_network
3. Wikipedia. Term *the Semantic Web*: http://en.wikipedia.org/wiki/Semantic_Web
4. *IRC channel of the FOAF developers*: Server irc:irc.freenode.net, channel *#foaf*
5. Application FOAF-a-Matic: http://www.ldodds.com/blog/archives/000087.html
6. Application *FOAF Explorer*: http://xml.mfd-consult.dk/foaf/explorer/
7. Application *FOAF Web View*: http://eikeon.com/foaf
8. Application *People Link.org*: http://beta.plink.org/, the project was shut down.
9. Application *Advanced FOAF Explorer* (AFE): http://foaf-explorer.zapisky.info/
10. Article "*Jednoduchý MVC framework napsaný v PHP"* (Simple MVC framework written in PHP) posted by Josef Petrák on 8th February 2006: http://zapisky.info/?item=jednoduchy-mvc-framework-napsany-v-php
11. Library *PHP API for PHP*: http://www.wiwiss.fu-berlin.de/suhl/bizer/rdfapi/
12. Specification of *the FOAF Vocabulary*: http://xmlns.com/foaf/0.1/
13. RDF Schema *RELATIONSHIP: A vocabulary for describing relationships between people*: http://vocab.org/relationship/.
14. RDF Primer, Frank Manola, Eric Miller, W3C Recommendation from 10th February 2004, The latest version is available at http://www.w3.org/TR/rdf-primer/.
15. OWL Web Ontology Language Reference, Dean M., Schreiber G (Editors); van Harmelen F., Hendler J., Horrocks I., McGuinness D.L., Patel-Schneider P.F., Stein L.A. (Authors), W3C Recommendation, 10 February 2004. The latest version is http://www.w3.org/TR/owl-ref/.
16. Extensible Markup Language (XML) 1.0, Second Edition, Bray T., Paoli J., Sperberg-McQueen C.M., Maler E. (Editors), World Wide Web Consortium, 6 October 2000. The latest version is http://www.w3.org/TR/REC-xml.
17. XHTML 1.0 The Extensible HyperText Markup Language (Second Edition), World Wide Web Consortium. 26 January 2000, revised 1 August 2002. The latest version of XHTML 1 is available at http://www.w3.org/TR/xhtml1/.
18. RDF Vocabulary Description Language 1.0: RDF Schema, Brickley D., Guha R.V. (Editors), W3C Recommendation, 10 February 2004. The latest version is http://www.w3.org/TR/rdf-schema/.