Proceedings of the 2nd International Workshop on

Scripting for the Semantic Web (SFSW 2006)

Co-located with 3rd European Semantic Web Conference
June 11-14, 2006

eswc06
Budva, Montenegro
11 - 14 June 2006
European Semantic Systems Initiative
3rd European Semantic Web Conference 2006

Workshop Co-Chairs' Message

SFSW 2006 - Workshop on Scripting for the Semantic Web

Scripting languages such as Python, PHP, Perl, JavaScript, Ruby, ASP, JSP and ActionScript are playing a central role in current development towards flexible, lightweight web applications following the AJAX and REST design paradigms. These languages are the tools of a generation of web programmers who use them to quickly create server and client-side web applications. Scripting languages are lightweight and easy to learn, but on the other hand mature enough to be used within complex applications. Many deployed Semantic Web applications from the wiki, blog, FOAF and RSS communities, as well as many innovative mashups from the Web 2.0 and Open Data movements are using scripting languages and it is likely that the process of RDF-izing existing database-backed websites, wikis, blogs and content management systems will largely rely on scripting languages.

The workshop brings together developers of the RDF base infrastructure for scripting languages with practitioners building applications using these languages. Last year's workshop in Hersonissos/Crete focused on giving an overview about the support for Semantic Web technologies within scripting languages. The special focus of this year's workshop is to showcase innovative Semantic Web applications relying on script languages and to give an overview about currently emerging Web 2.0 mashups and their interrelations with the Semantic Web. Hence, this year's workshop includes a scripting challenge, awarding a price to the most innovative scripting application.

We would like to thank the organizers of ESWC conference for supporting the workshop. We especially thank all members of the SFSW program committee for providing their expertise and giving elaborate feedback to the authors. Last but not least, we hope that you will enjoy the workshop and the whole conference.

Sören Auer, Universität Leipzig, Germany
Chris Bizer, Freie Universität Berlin, Germany
Libby Miller, @Semantics, Italy

SFSW 2006 Program Committee

- Danny Ayers, Independent Author, Italy
- Dave Beckett, Yahoo!, USA
- Matt Biddulph, Independent Developer, United Kingdom
- Dan Brickley, Semantic Web Vapourware, United Kingdom
- Stefan Decker, DERI, Ireland
- Edd Dumbill, Useful Information Company, United Kingdom
- Leigh Dodds, Ingenta, United Kingdom
- Klaus-Peter Fähnrich, Universität Leipzig, Germany
- Morten Frederiksen, MFD Consult, Denmark
- Chris Goad, Map Bureau, United States
- Gunnar AA. Grimnes, DFKI, Germany
- Frank Fuchs-Kittowski, FhG - ISST, Germany
- Daniel Krech, University of Maryland, United States
- Jim Ley, Independent Developer, United Kingdom
- Lutz Maicher, Universität Leipzig, Germany
- Benjamin Nowack, appmosphere web applications, Germany
- Uche Ogbuji, Fourthought, United States
- Sean Palmer, Independent Developer, United Kingdom
- Alberto Reggiori, @Semantics, Italy
- Guus Schreiber, Free University Amsterdam, Netherlands
- Robert Tolksdorf, Freie Universität Berlin, Germany
- Giovanni Tummarello, Universita' Politenica delle Marche, Italy

Table of Contents

Semantic Scripting Challenge Submissions

Deep Integration of Python with Web Ontology Language

Marian Babik, Ladislav Hluchy *

Intelligent and Knowledge-based Technologies Group,
Department of Parallel and Distributed Computing, Institute of Informatics,
Slovak Academy of Sciences
Marian.Babik@saske.sk, Ladislav.Hluchy@savba.sk

Abstract. The Semantic Web is a vision for the future of the Web in which information is given explicit meaning, making it easier for machines to automatically process and integrate information available on the Web. Semantic Web will build on the well known Semantic Web language stack, part of which is the Web Ontology Language (OWL). Python is an interpreted, object-oriented, extensible programming language, which provides an excellent combination of clarity and versatility. The deep integration of both languages is one of the novel approaches for enabling free and interoperable data [1].

In this article we present a metaclass-based implementation of the deep integration ideas. The implementation is an early Python prototype supporting in-line class and properties declaration, instance creation and simple triple-based queries. The implementation is backed up by a well known OWL-DL reasoner Pellet [3]. The integration of the Python and OWL-DL through meta-class programming provides a unique approach, which can be implemented with any metaclass enabled scripting language.

1 Introduction

The deep integration of scripting languages and Semantic Web has introduced an idea of importing the ontologies directly into the programming context so that its classes are usable alongside classes defined normally. This can provide a more natural mapping of OWL-DL than classic APIs, reflecting the set-theoretic semantics of OWL-DL, while preserving the access to the classic Python objects. Such integration also encourages separation of concerns among declarative and procedural and encourages a new wave of programming, where problems can be

* Acknowledgments: The research reported in this paper has been partially financed by the EU within the project IST-2004-511385 K-WfGrid and Slovak national projects, Research and development of a knowledge based system to support workflow management in organizations with administrative processes, APVT-51-024604; Tools for acquisition, organization and maintenance of knowledge in an environment of heterogeneous information resources, SPVV 1025/04; Efficient tools and mechanisms for grid computing (2006-2008) VEGA 2/6103/6.

defined by using description logics [10] and manipulated by dynamic scripting languages [1]. The approach represents a unification, that allows both languages to be conveniently used for different subproblems in the software-engineering environment.

In this article we would like to introduce an early prototype, which implements some of the ideas of the deep integration in Python language [8]. It supports in-line declaration of OWL classes and properties, instance creation and simple triple-based queries [2]. We will emphasize the notion of modeling intensional sets (i-sets) through metaclasses. We will also discuss the possible drawbacks of the approach and the current implementation.

2 Intensional Sets and Metaclasses

Intensional sets as introduced in [1] are sets that are described with OWL DL's construct and according to this description, encompass all fitting instances. A sample intensional set can be defined by using Notation3 (N3) [12] as, e.g. ":Person a owl:Class; rdfs:subClassOf :Mortal". This simply states that Person is also a Mortal. Assuming we introduce two instances, e.g. ":John a :Person and :Jane a :Mortal", the instances of Mortal are both John and Jane. Please note, that N3 is used only for demonstration purposes, the current implementation can also support NTriples and RDF/XML.

Terminology-wise, a metaclass is simply "the class of a class". Any class whose instances are themselves classes, is a metaclass. A metaclass-based implementation of the intensional sets is based on the core metaclass Thing, whose constructor accepts two main attributes, i.e. default namespace and N3 description of the i-set. The instance of the metaclass is then a mapping of the OWL class to the intensional set. Following the above example class Person can be created with a Python construct:

```
Person = Thing('Person', (),
{defined_by: 'a owl:Class; rdfs:subClassOf :Mortal',\
 namespace: 'http://samplens.org/test#'})
```

This creates a Python class representing the intensional set for Person and its namespace. In the background it also updates the knowledge base with the new assertion. The individual John can then be instantiated simply by calling $John = Person()$. This statement calls the default constructor of the class Person, which provides support for asserting new OWL individual into the knowledge base. A similar metaclass is used for the OWL property except that it can not be instantiated. The constructor is used here for different purpose, i.e. to create a relation between classes or individuals. The notion of importing the ontology into the Python's namespace is then a matter of decomposing the ontology into the groups of intensional sets, generating Python classes for these sets and creating the instances.

This kind of programming is also called metaclass programming and can provide a transparent way how to generate new OWL classes and properties. Since metaclasses act like regular classes it is possible to extend their functionality by inheriting from the base metaclass. It is also simple to hide the complex tasks needed for accessing the knowledge base, reasoner and processing the mappings between OWL-DL's concepts and their respective Python counterparts.

3 Sample session

A sample session shows an in-line declaration of a class *Person*. This is implemented by calling a static method *new* of the metaclass *Thing* (the static method *new* is used to hide the complex call introduced in Sec. 2). An instance is created by calling the *Person's* constructor, which in fact creates an in-line declaration of the OWL individual (*John*). The print statements show the Python's view of the respective objects, i.e. *Person* as a class and *John* as an instance of the class *Person*. We also show a more complex definition of the *PersonWithSingleSon*, which we will later use to demonstrate the reasoning about property assertions.

```
>>> from Thing import Thing
>>> Person = Thing.new(’Person’,’ a owl:Class .’)
>>> John = Person()
>>> print Person
<class ’Thing.Person’>
>>> print John
<Thing.Person object at 0xb7d0b50c>
>>> PersonWithSingleSon = Thing.new(’PersonWithSingleSon’, \
    """ a  owl:Class ; rdfs:subClassOf   [ a  owl:Restriction ;
     owl:cardinality "1"^^<http://www.w3.org/2001/XMLSchema#int> ;
     owl:onProperty :hasSon
                                         ] ;
     rdfs:subClassOf  [ a  owl:Restriction ;
     owl:cardinality "1"^^<http://www.w3.org/2001/XMLSchema#int> ;
     owl:onProperty :hasChild ] .""")
```

A similar way can be used for in-line declarations of OWL properties. Compared to a class declaration the returned Python class can not be instantiated (i.e. returns *None*).

```
>>> from PropertyThing import Property
>>> hasChild = Property.new(’hasChild’,’ a owl:ObjectProperty .’)
>>> print hasChild
<class ’PropertyThing.hasChild’>
>>> hasSon = Property.new(’hasSon’, ’ a owl:ObjectProperty ;
rdfs:subPropertyOf :hasChild .’)
```

Properties are naturally used to assign relationships between OWL classes or individuals, which can be as simple as calling:

```
>>> Bob = PersonWithSingleSon()
>>> hasChild(Bob, John)
```

Assuming we have declared several instances of the class *Person* we can find them by iterating over the class list. It is also possible to ask any triple like queries (the query shown also demonstrates reasoning about the property assertions).

```
>>> for individual in Person.findInstances():
...     print individual, individual.name
<Thing.Man object at 0xb7d0b64c> Peter
<Thing.Person object at 0xb7d0b50c> John
<Thing.Person object at 0xb7d0b6ec> Jane

>>> for who in hasSon.query(Bob):
...     who.name
'John'
>>> print hasSon.query(Bob, John)
1
```

4 Implementation and Drawbacks

Apart from the metaclass-based implementation of the i-sets, it is necessary to support query answering, i.e. provide an interface to the OWL-DL reasoner. There are several choices for the OWL-DL reasoners including Racer, Pellet and Kaon2 [4, 3, 19]. Although these reasoners provide sufficient support for OWL-DL their integration with Python is not trivial since they are mostly based on Java (except Racer) and although they can work in server-like mode Python's support for standard protocols (like DIG) is missing. Other possibilities are to use Python-based inference engines like CWM, Pychinko or Euler [13–15]. However, due to the performance reasons, lack of documentation or too early prototypes we have decided to use Java-based reasoners. We have managed to successfully use JPype [16], which interfaces Java and Python at native level of virtual machines (using JNI). This enables the possibility to access Java libraries from within CPython. Having the ability to call Java and use all the capabilities of the current version of CPython (e.g. metaclasses) is a big advantage over the other approaches such as Jython or JPE [11, 9]. We have developed a wrapper class, which can call Jena and Pellet APIs and perform the needed reasoning [5, 3]. The wrapper class is implemented as a singleton and interfaces the Python calls to the reasoner and RDF/OWL API and forwards it to the JVM with the help of the JPype.

One of the main drawbacks of the current implementation is the fact, that it doesn't support open world semantics (OWA). Although the reasoner in the current implementation can perform OWA reasoning and thus it is possible to correctly answer queries, the Python's semantics are based on the boolean values. One of the possibilities is to use epistemic operator as suggested in [1], however this is yet to be implemented. Another problem when dealing with ontologies are namespaces. In the current prototype we have added a set of namespaces that constitute the corresponding OWL class or property description as an attribute of the Python's class. This attribute can then be used to generate the headers for the N3 description. This approach needs further extension to support management of different ontologies. One of the possibilities would be to re-use Python's module namespace by introducing a core ontology class. This ontology class would serve as a default namespace handler as well as a common importing point for the ontology classes.

The other drawback of the approach is the performance of the reasoner, which is due to the nature of the JPype implementation (the conversions between virtual machines imposes rather large performance bottlenecks). This can be solved by extending the support for other Python to Java APIs and possible implementation of specialized client-server protocols.

5 Related Work

To our best knowledge there is currently no Python implementation of the deep integration ideas. There is a Ruby implementation done by Obie Fernandez, however it is not known what is the current status of the project [20].

The most popular existing OWL APIs, that provide programmatic access to the OWL structures are based on Java [5, 7]. Since Java is a frame language its notion of polymorphism is very different than in RDF/OWL. This is usually solved by incorporating design patterns, which make the APIs quite complex and sometimes difficult to use. The dynamic nature of the scripting languages can support OWL/RDF level of polymorphism and thus it is possible to directly expose the OWL structures as Python classes without any API interfaces. One of the interesting Java projects, which tries to automatically map OWL ontologies into Java through Java Beans is based on the ideas shown in [17]. This approach tries to find a way how to directly map the ontologies to the hierarchy of the Java classes and interfaces.

Among the existing Python libraries, which support RDF and OWL, the most interesting in terms of partial integration are Sparta and Tramp, which bind RDF graph nodes to Python objects and RDF arcs to attributes of such objects [21, 6]. The projects however doesn't clearly address the OWL and are mainly considered with RDF. It is thus difficult to evaluate what is the level of support for the inferred OWL models.

6 Conclusion

We have described a metaclass-based prototype implementation of the deep integration ideas. We have discussed the advantages and shortcomings of the current implementation. We would like to note, that this a work in progress, which is constantly changing and this is just a report of the current status. At the time of writing authors are setting up an open source project, which will host the implementation of the ideas presented. There are many other open questions, that we haven't covered here including integration of query languages (possibility to re-use ideas from native queries [18]); serialization of the ontologies; representation of rules, concrete domains, etc. We hope that having an initial implementation is a good start and that its continuation will contribute to the success of the deep integration of the scripting and Semantic Web.

References

1. Vrandecic, D., Deep Integration of Scripting Languages and Semantic Web Technologies, In Soren Auer, Chris Bizer, Libby Miller, 1st International Workshop on Scripting for the Semantic Web SFSW 2005 , volume 135 of CEUR Workshop Proceedings. CEUR-WS.org, Herakleion, Greece, May 2005. ISSN: 1613-0073
2. Web Ontology Language (OWL), see http://www.w3.org/TR/owl-features/
3. Pellet OWL Reasoner, see http://www.mindswap.org/2003/pellet/index.shtml
4. RacerPro Reasoner, see http://www.racer-systems.com
5. Jena: A Semantic Web Framework for Java, see http://www.hpl.hp.com/semweb/jena2.htm.
6. TRAMP: Makes RDF look like Python data structures http://www.aaronsw.com/2002/tramp, http://www.amk.ca/conceit/rdf-interface.html
7. Bechhofer, S., Lord, P., Volz,R.:Cooking the Semantic Web with the OWL API. 2nd International Semantic Web Conference, ISWC, Sanibel Island, Florida, October 2003
8. G. van Rossum, Computer programming for everybody. Technical report, Corporation for National Research Initiatives, 1999
9. Java-Python Extension, http://sourceforge.net/projects/jpe
10. Baader, F., Calvanese, D., McGuinness, D.,L., Nardi, D. and Patel-Schneider,P., F. editors. The description logic handbook: theory, implementation, and applications. Cambridge University Press, New York, NY, USA, 2003.
11. Jython, Java implementation of the Python, http://www.jython.org/
12. Notation 3, see http://www.w3.org/DesignIssues/Notation3.html
13. Closed World Machine, see http://www.w3.org/2000/10/swap/doc/cwm.html
14. Katz, Y., Clark, K. and Parsia, B., Pychinko: A native python rule engine. In International Python Conference 05, 2005.
15. Euler proof mechanism, see http://www.agfa.com/w3c/euler/
16. JPype, Java to Python integration, see http://jpype.sourceforge.net/
17. Kalyanpur, A., Pastor, D., Battle, S. and Padget, J., Automatic mapping of owl ontologies into java. In Proceedings of Software Engg. - Knowledge Engg. (SEKE) 2004, Banff, Canada, June 2004.
18. Cook, W. R. and Rosenberger, C., Native Queries for Persistent Objects, Dr. Dobb's Journal, February 2006
19. Hustadt, U., Motik, B., Sattler, U.. Reducing SHIQ Description Logic to Disjunctive Datalog Programs. Proc. of the 9th International Conference on Knowledge Representation and Reasoning (KR2004), June 2004, Whistler, Canada, pp. 152-16
20. Fernandez, O., Deep Integration of Ruby with Semantic Web Ontologies, see gigaton.thoughtworks.net/ ofernand1/DeepIntegration.pdf
21. Sparta, Python API for RDF, see http://www.mnot.net/sw/sparta/

ActiveRDF: object-oriented RDF in Ruby*

Eyal Oren and Renaud Delbru

DERI Galway
firstname.lastname@deri.org

Abstract. Although most developers are object-oriented, programming RDF is triple-oriented. Bridging this gap, by developing a truly object-oriented API that uses domain terminology, is not straightforward, because of the dynamic and semi-structured nature of RDF and the open-world semantics of RDF Schema.
We present ActiveRDF, our object-oriented library for accessing RDF data. ActiveRDF is completely dynamic, offers full manipulation and querying of RDF data, does not rely on a schema and can be used against different data-stores. In addition, the integration with the popular Rails framework enables very easy development of Semantic Web applications.

1 Introduction

The Semantic Web is a web of data that can be processed by machines, enabling them to interpret, combine and use Web data [1, p. 191]. RDF[1] is one of the foundations of the Semantic Web. A statement in RDF is a triple stating that a subject has a property with some value.

Programming in the Semantic Web means programming against RDF data. And although most current developers have an object-oriented attitude, programming in RDF is currently triple-based, and getting from the one to the other is cumbersome.

The development of an object-oriented RDF API has been suggested several times [2,7,8], but developing such an API faces several challenges. Using a statically typed and compiled language like Java does not address the challenges correctly (as explained next). A scripting language such as Ruby on the other hand, allows us to fully address these challenges and develop ActiveRDF[2], our "deeply integrated" [8] object-oriented RDF API.

1.1 Overview

The following example summarises ActiveRDF and its features. The example program connects to a YARS[3] RDF database, creates a person and saves that person into the database:

* This material is based upon works supported by the Science Foundation Ireland under Grants No. SFI/02/CE1/I131 and SFI/04/BR/CS0694.

[1] http://w3.org/RDF/

[2] See http://activerdf.m3pe.org for download and documentation (open source).

[3] http://sw.deri.org/2004/06/yars/

```
1  class Person < IdentifiedResource
2    set_class_uri 'http://xmlns.com/foaf/0.1/Person'
3  end
4
5  NodeFactory.connection(:adapter => :yars, :host => 'm3pe.org')
6
7  renaud = Person.create 'http://activerdf.m3pe.org/renaud'
8  eyal = Person.find_by_firstName 'eyal'
9  renaud.firstName = 'Renaud'
10 renaud.lastName = 'Delbru'
11 renaud.knows = eyal
12 renaud.save
```

The main features of ActiveRDF are the following (Tramp[4] is a comparable dynamic API, but has only the two first features):

1. An RDF manipulation language (domain-specific language) using the terminology from the dataset, e.g. offering `renaud.firstName` in line 9 instead of `Resource.getProperty`.
2. Read and write access to arbitrary RDF data, exposing data as objects, and translating all method invocations on those objects as RDF queries, e.g. `renaud.firstName` and `renaud.save` in line 12.
3. Usage of various data-stores through adapter system, indicated in line 5 with `:adapter => :yars`.
4. Dynamic query methods based on the data properties, for example the `find_by_firstName` in line 8.
5. Data-schema independence: classes, objects, and methods are created on-the-fly from the apparent structure in the data (currently using the URI definition given in `class Person` on line 1–3).
6. Object caching that optimises performance and preserves memory through lazy data fetching.
7. Integration with Rails[5], a popular web application framework for Ruby, putting the Semantic Web on Rails.

1.2 Outline

The rest of the paper proceeds as follows: in Sect. 2 we discuss the challenges in designing an object-oriented RDF API and argue that a dynamic scripting language (as opposed to a static language) is very suitable for such an API. We demonstrate the usage of ActiveRDF in more detail in Sect. 3 and describe the internal architecture in Sect. 4. In Sect. 5 we describe how we used ActiveRDF to create a Web application for browsing arbitrary RDF data with little effort, and we conclude in Sect. 6.

[4] `http://www.aaronsw.com/2002/tramp`

[5] `http://rubyonrails.org`

2 Background

In this section, we introduce scripting languages, describe the challenges that need to be addressed to develop an object-oriented RDF API, and explain why a scripting language such as Ruby is well suited for such an API.

2.1 Scripting languages

There is no exact definition of "scripting languages", but we can generally characterise them as high-level programming languages, less efficient but more flexible than compiled languages [6]:

Interpreted Scripting languages are usually interpreted instead of compiled, allowing quick turnaround development and making applications more flexible through runtime programming.

Dynamic typing Scripting languages are usually weakly typed, without prior restrictions on how a piece of data can be used. Ruby for example has the "duck-typing" mechanism in which object types are determined by their runtime capabilities instead[6] of by their class definition.

Meta-Programming Scripting languages usually do not strongly separate data and code, and allow code to be created, changed, and added during runtime. In Ruby, it is possible to change the behaviour of all objects during runtime and for example add code to a single object (without changing its class).

Reflection Scripting languages are usually suited for flexible integration tasks and are supposed to be used in dynamic environments. Scripting languages usually allow strong reflection (the possibility to easily investigate data and code during runtime) and runtime interrogation of objects instead of relying on their class definitions.

The flexibility of scripting languages (as opposed to statically-typed and compiled languages) allows us to develop a truly object-oriented RDF API.

2.2 Challenges in object-oriented RDF

There are many RDF APIs available in various different programming languages[7], most of them for accessing one specific RDF data-store. Most RDF APIs, such as in Sesame[8], Jena[9] or Redland[10] offer only generic methods such as `getStatement, getResource, getProperty, getObject`.

Using these generic APIs is quite cumbersome, and a more usable API has been advocated several times [2,7,8]. Such an API would map RDF resources to

[6] Ruby also has the strong object-oriented notion of (defined) classes, but the more dynamic notion of duck-typing is preferred.

[7] `http://www.wiwiss.fu-berlin.de/suhl/bizer/toolkits`

[8] `http://openrdf.org`

[9] `http://jena.sourceforge.net`

[10] `http://librdf.org`

programming objects, RDF predicates to methods on those objects, and possibly RDF Schema[11] classes to programming classes. This API would for example not contain `Resource.getProperty` but `Person.getFirstName`.

However, providing such an object-oriented API for RDF data is not straightforward, given the following issues:

Type system The semantics of classes and instances in (description-logic based) RDF Schema and the semantics of (constraint-based) object-oriented type systems differ fundamentally [4].

Semi-structured data RDF data are semi-structured, may appear without any schema information and may be untyped. In object-oriented type systems, all objects must have a type and the type defines their properties.

Inheritance RDF Schema allows instances to inherit from multiple classes (multi-inheritance), but many object-oriented type systems only allow single inheritance.

Flexibility RDF is designed for integration of heterogeneous data with varying structure. Even if RDF schemas (or richer ontologies) are used to describe the data, these schemas may well evolve and should not be expected to be stable. An application that uses RDF data should be flexible and not depend on a static RDF Schema.

Given these issues, we investigate the suitability of scripting languages for RDF data.

2.3 Addressing these challenges with a dynamic language

The development of an object-oriented API has been attempted using a statically-typed language (Java) in RdfReactor[12], Elmo[13] and Jastor[14]. These approaches ignore the flexible and semi-structured nature of RDF data and instead:

1. assume the existence of a schema, because they rely on the RDF Schema to generate corresponding classes,
2. assume the stability of the schema, because they require manual regeneration and recompilation if the schema changes and
3. assume the conformance of RDF data to such a schema, because they do not allow objects with different structure than their class definition.

Unfortunately, these three assumptions are generally wrong, and severely restrict the usage of RDF. A dynamic scripting language on the other hand is very well suited for exposing RDF data and allows us to address the above issues[15]:

[11] `http://www.w3.org/TR/rdf-schema/`

[12] `http://rdfreactor.ontoware.org/`

[13] `http://www.openrdf.org/doc/elmo/users/index.html`

[14] `http://jastor.sourceforge.net/`

[15] We do not claim that compiled languages cannot address these challenges (they are after all Turing complete), but that scripting languages are especially suited and address all these issues very easily.

Type system Scripting languages have a dynamic type system in which objects can have no type or multiple types (although not necessarily at one-time). Types are not defined prior but determined at runtime by the capabilities of an object.

Semi-Structured data Again, the dynamic type system in scripting languages does not require objects to have exactly one type during their lifetime and does not limit object functionality to their defined type. For example, the Ruby “mixin” mechanism allows us to extend or override objects and classes with specific functionality and data at runtime.

Inheritance Most scripting languages only allow single inheritance, but their meta-programming capabilities allow us to either i) override their internal type system, or ii) generate a compatible single-inheritance class hierarchy on-the-fly during runtime.

Flexibility Scripting languages are interpreted and thus do not require compilation. This allows us to generate a *virtual* API on the fly, during runtime. Changes in the data schema do not require regeneration and recompilation of the API, but are immediately accounted for. To use such a flexible virtual API the application needs to employ reflection (or introspection) at runtime to discover the currently available classes and their functionality.

In summary, dynamic scripting languages offer us exactly those properties to offer a virtual and flexible API for RDF data. Our arguments apply equally well to any dynamic language with these capabilities. We have chosen Ruby as a simple yet powerful scripting language, which in addition allows us to leverage the popular Rails framework for easy development of complete Semantic Web applications.

3 Usage

We now show the most salient ActiveRDF features with two examples. Please refer to the manual[16] for more information on the usage of ActiveRDF. Sect. 4 and will explain how the features are implemented.

3.1 Create, read, update and delete

ActiveRDF maps RDF resources to Ruby objects and RDF properties to methods (attributes) on these objects. If a schema is defined, we also map RDF Schema classes to Ruby classes and map predicates to class methods. The default mapping uses the local part of the schema classes to construct the Ruby classes; the defaults can be overridden to prevent naming clashes (e.g. `foaf:name` could be mapped to `FoafName` and `doap:name` to `DoapName`).

If no schema is defined, we inspect the data and map resource predicates to object properties directly, e.g. if only the triple `:eyal :eats "food"` is available,

[16] `http://activerdf.org/manual/`

we create a Ruby object for `eyal` and add the method `eats` to this object (not to its class).

For objects with cardinality larger than one, we automatically construct an array with the constituent values; we do not (yet) support RDF lists and collections.

Creating objects either loads an existing resource or creates a new resource. The following example shows how to load an existing resource, interrogate its capabilities, read and change one of its properties, and save the changes back. The last part shows how to use standard Ruby closure to print the name of each of Renaud's friends.

```
renaud = Person.create('http://activerdf.m3pe.org/renaud')
renaud.methods         ... ['firstName', 'lastName', 'knows', ...]
renaud.firstName       ... 'renaud'
renaud.firstName = 'Renaud'
renaud.save

renaud.knows.each do |friend|
  puts friend.firstName
end
```

3.2 Dynamic finders

ActiveRDF provides dynamic search methods based on the runtime capabilities of objects. We can use these dynamic search methods to find particular RDF data; the search methods are automatically translated into queries on the dataset.

The following example shows how to use `Person.find_by_firstName` and `Person.find_by_knows` to find some resources. Finders are available for all combinations of object predicates, and can either search for exact matches or for keyword matches (if the underlying data-store supports keyword search).

```
eyal = Person.find_by_firstName 'Eyal'
renaud = Person.find_by_knows eyal
all_johns = Person.find_by_keyword_name 'john'
other = Person.find_by_keyword_name_and_age 'jack', '30'
```

4 Architecture

In this section, we give a brief overview of the architecture of ActiveRDF. ActiveRDF follows ActiveRecord pattern [3, p. 160] which abstracts the database, simplifies data access and ensures data consistency, but adjusted for RDF data.

4.1 Overview

ActiveRDF is composed in four layers, shown in Fig. 1:

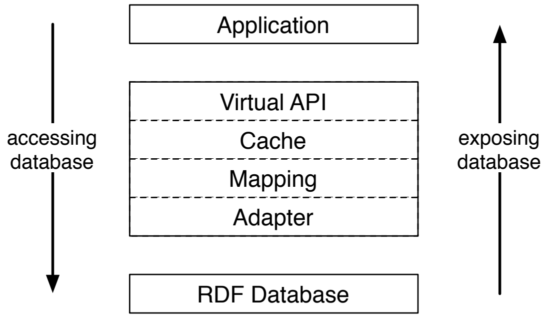


Fig. 1. Overview of the ActiveRDF architecture

Virtual API The virtual API is the application entry point and provides all the ActiveRDF functionality: it provides the domain model with all its manipulation methods and generic search methods. It is not a generated API (hence the name *virtual*) but uses Ruby meta-programming to catch unhandled method calls (such as `renaud.firstName`) and respond to them.

Cache A caching mechanism can be used to reduce access to the database and improve time performance (in exchange for memory).

Mapping Maps RDF data to Ruby objects and data manipulation to Ruby methods. For example, when the application calls a find method or when we create a new person, the mapping layer translates this operation (using an adapter) into a specific query on the data-store. The mapping layer also creates Ruby classes from RDF Schema classes (with methods), or adds Ruby objects (with their own methods) if no schema is available.

Adapter Provides access to a specific RDF data-store by translating generic RDF operations to a store-specific API[17]. The adapter layer enables communication with a specific RDF data-store. Each adapter offers a simple low-level API (consisting of *query, add, remove* and *save*) which is used by the mapping layer. The low-level API is purposely kept simple, so that new adapters can be easily added. The real mapping logic is provided by the generic mapping layer.

4.2 Feature implementation

We now briefly explain how the features mentioned in Sect. 1.1 are implemented in the architecture:

1. The virtual API offers a RDF manipulation language (using Ruby meta-programming) which uses the mapping layer to create classes, objects, and methods which respect the terminology of the RDF data.

[17] in absence of general standardised query language that provides create, read, update, and delete access to RDF data-stores.

2. The virtual API offers read and write access and uses the mapping layer to translate operations into RDF queries.
3. Adapters provide access to various data-stores and new adapters can easily be added since they require only little code.
4. The virtual API offers dynamic search methods (using Ruby reflection) and uses the mapping layer to translate searches into RDF queries.
5. The mapping layer is completely dynamic and maps RDF Schema classes and properties to Ruby classes and methods. In the absence of a schema the mapping layer infers properties of *instances* and adds it to the corresponding objects directly achieving data-schema independence.
6. The caching layer (if enabled) minimises database communication by keeping loaded RDF resources in memory and ensures data consistency by making sure not more than one copy of an RDF resource is created.
7. The implementation design ensures integration with Rails: we have created ActiveRDF to be API-compatible with the Rails framework.

The functionality of ActiveRDF (especially if combined with Rails) allows rapid development of Semantic Web applications that fully respect the principles of RDF.

5 Case Study

We have not yet performed an extensive evaluation of the usability and improved productivity of ActiveRDF (compared to common RDF APIs). Instead we report some anecdotal evidence in our own development of a Web applications that uses ActiveRDF in combination with Rails.

5.1 Semantic Web with Ruby on Rails

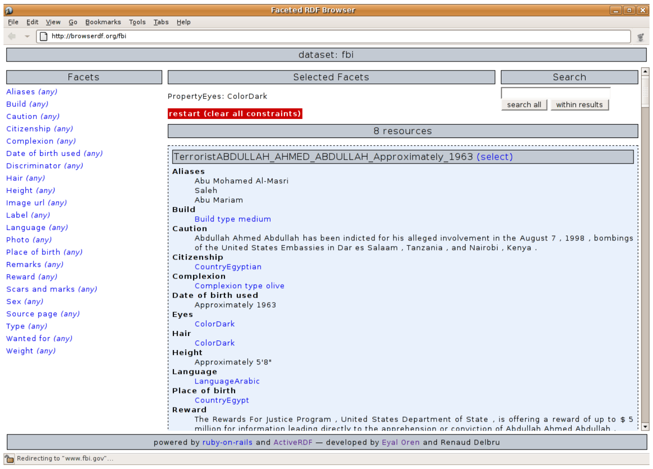
Rails is a RAD (rapid application development) framework for web applications. It follows the model-view-controller paradigm. The basic framework of Rails allows programmers to quickly populate this paradigm with their domain: the model is (usually) provided by an existing database, the view consists of HTML pages with embedded Ruby code, and the controller is some simple Ruby code.

Rails assumes that most web applications are built on databases (the web application offers a view on the database and operations on that view) and makes that relation as easy as possible. Using ActiveRecord, Rails uses database tables as models, offering database tuples as instances in the Ruby environment.

ActiveRDF can serve as data layer in Rails. Two data layers currently exist for Rails: ActiveRecord provides a mapping from relational databases to Ruby objects and ActiveLDAP provides a mapping from LDAP resources to Ruby objects. ActiveRDF can serve as an alternative data layer in Rails, allowing rapid development of semantic web applications using the Rails framework.

5.2 Building a faceted RDF browser

We have used ActiveRDF in the development of our prototype faceted browser, available on `http://browserdf.org` and shown in Fig. 2. We have improved faceted browsing, a navigation technique for structured data, with automatic facet construction to enable browsing of semi-structured data [5].



Aliases (any)
Build (any)
Caution (any)
Citizenship (any)
Complexion (any)
Date of birth used (any)
Discriminator (any)
Hair (any)
Height (any)
Image url (any)
Label (any)
Language (any)
Photo (any)
Place of birth (any)
Remarks (any)
Reward (any)
Scars and marks (any)
Sex (any)
Source page (any)
Type (any)
Wanted for (any)
Weight (any)

PropertyEyes: ColorDark

TerroristABDULLAH_AHMED_ABDULLAH_Approximately_1963 (select)

Aliases
Abu Mohamed Al-Masri
Saleh
Abu Mariam
Build
Build type medium
Caution
Abdullah Ahmed Abdullah has been indicted for his alleged involvement in the August 7 , 1998 , bombings of the United States Embassies in Dar es Salaam , Tanzania , and Nairobi , Kenya .
Citizenship
CountryEgyptian
Complexion
Complexion type olive
Date of birth used
Approximately 1963
Eyes
ColorDark
Hair
ColorDark
Height
Approximately 5'8"
Language
LanguageArabic
Place of birth
CountryEgypt
Reward
The Rewards For Justice Program , United States Department of State , is offering a reward of up to $ 5 million for information leading directly to the apprehension or conviction of Abdullah Ahmed Abdullah .

powered by ruby-on-rails and ActiveRDF — developed by Eyal Oren and Renaud Delbru

Redirecting to "www.fbi.gov"...

Fig. 2. Faceted browsing prototype

The prototype is implemented in Ruby, using ActiveRDF and Rails. After finishing the theoretical work on the automated facet construction, the application was developed in only several days. In total, there are around 385 lines of code: 250 lines for the controller (consisting mostly of the facet computation algorithm), 35 lines for the data model and 100 lines for the interface which includes all RDF manipulations for display.

6 Conclusion

ActiveRDF is an object-oriented library for RDF data. It can be used with arbitrary data-stores and offers full manipulation of RDF through a virtual API that respects the data terminology. ActiveRDF is completely dynamic and can work without any schema information. Additionally, the integration with the popular Rails framework enables very easy development of Semantic Web applications.

We have analysed the problems of object-oriented access to RDF data and shown that a scripting language is well suited for such a task. In ActiveRDF, we address the challenges as follows:

1. the mapping layer does not rely on the existence of a schema, but can also infer properties from the instance data (as shown in Sect. 1.1),

2. the virtual API is provided dynamically and automatically modified during runtime to stay consistent with a dynamic schema.
3. the mapping layer does not assume the conformance of RDF data to a schema, but instead allows objects with other capabilities than their class definition.

6.1 Discussion

ActiveRDF is by design restricted to direct data manipulation; we do not perform any reasoning, validation, or constraint maintenance. In our opinion, those are tasks for the RDF store, similar to consistency maintenance in databases.

One could argue that statically generated classes have one advantage: they result in readable APIs, that people can program against. In our opinion, that is not a viable philosophy on the Semantic Web. Instead of expecting a static API one should anticipate various data and introspect it at runtime. On the other hand, static APIs allow code-completion, but that could technically be done with virtual APIs as well (using introspection during programming).

6.2 Future work

We have currently released the first version of ActiveRDF and are continuing to improve it. We outline some important issues and how we intend to resolve them. First, multiple inheritance is not yet implemented in the first release, but we are already working on a technique that overrides the internal Ruby type-system, manages our own types and allows multiple inheritance. Secondly, we are removing the cumbersome and technically unnecessary need to pre-define all classes (as shown in the first example in Sect. 1.1). And finally, we plan to do an usage evaluation of ActiveRDF versus other RDF APIs, on (data querying) perfomance and on programmer productivity; we reserve such an evaluation for future work.

References

1. T. Berners-Lee. *Weaving the Web – The Past, Present and Future of the World Wide Web by its Inventor.* Texere, 2000.
2. O. Fernandez. Deep integration of ruby with semantic web ontologies. `http://gigaton.thoughtworks.net/~ofernand1/DeepIntegration.pdf`.
3. M. Fowler. *Patterns of Enterprise Application Architecture.* Addison-Wesley, 2002.
4. A. Kalyanpur, D. Pastor, S. Battle, and J. Padget. Automatic mapping of owl ontologies into java. In *SEKE.* 2004.
5. E. Oren, *et al.* Annotation and navigation in semantic wikis. In *SemWiki in ESWC.* 2006.
6. J. K. Ousterhout. Scripting: Higher-level programming for the 21st century. *IEEE Computer*, 31(3):23–30, 1998.
7. D. Schwabe, D. Brauner, D. A. Nunes, and G. Mamede. Hypersd: a semantic desktop as a semantic web application. In *SemDesk in ISWC.* 2005.
8. D. Vrandečić. Deep integration of scripting language and semantic web technologies. In *Scripting for the Semantic Web.* 2005.

Access Control on RDF Triple Stores from a Semantic Wiki Perspective

Sebastian Dietzold[1] and Sören Auer[1,2]

[1]Universität Leipzig
Department of Computer Science
{dietzold|auer}@informatik.uni-leipzig.de

[2]University of Pennsylvania
Department of Computer and Information Science
auer@seas.upenn.edu

Abstract. RDF triple stores are used to store and query large RDF models. Semantic Web applications built on top of such triple stores require methods allowing high-performance access control not restricted to per model directives. For the growing number of lightweight, scripted Semantic Web applications it is crucial to rely on access control methods which maintain a balance between expressiveness, simplicity and scalability. Starting from a Semantic Wiki application scenario we collect requirements for useful access control methods provided by the triple store. We derive a basic model for triple store access according to these requirements and review existing approaches in the field of policy management with regard to the requirements. Finally, a lightweight access control framework based on rule-controlled query filters is described.

Introduction

The efficient implementation of a Semantic Web application particularly depends on the underlying RDF API and triple store. Today's RDF triple stores are mostly build upon relational database management systems with dedicated database schemata and corresponding API methods which rewrite knowledge base queries into database queries (e.g. [1] and [2]). However, access control at the level of the underlying relational database lacks granularity if based on the entities table, row and database. There exist APIs which do not depend on a relational database (e.g. Redland [3] and Sesame [4]) but also when basing on them, the Semantic Web application has to establish access control mechanisms on their own.

We are convinced, that future RDF triple stores will be used as backends for application systems in analogy to existing relational databases. Assuming this, it is important to develop access control methods which do rely on the RDF data model and enable access controll with respect to metamodels based on the RDF data model, such as RDF-Schema and the different OWL flavors. A longer term aim is to make such methods integral part of future triple stores.

To have a more solid starting point for the formulation of requirements we selected the application scenario of Semantic Wikis. We define a Semantic Wiki as a collaborative software for modifying a shared knowledge base. Further, we assume this knowledge base to be an RDF graph which consists of RDF triples. A Semantic Wiki supports the collaborative process of instance acquisition and curation with respect to an often fuzzy and not well defined goal. This definition does not make any assumptions about the user frontend, because these wiki design principles are defined in [5] and are independent of the prefix "semantic".

1 Requirements on Access Control for a Semantic Wiki RDF Triple Store

In a 'classic' wiki we are used to think in access categories like account and page: an account has certain rights with respect to a page[1]. The specific rights which can be granted or revoked are typically the rights to read, modify, delete and annotate the page and to grant such rights for this page to some other account. Further, pages can be arranged in a page tree, which makes the application of access control rules on a page subtree necessary.

For a Semantic Wiki, the base entities we have to consider are not accounts and pages but sets of triples. Also, accounts and annotations can be identified using URI references and information about them represented as triples. Hence, access control should work on the granularity level of triples as well as on higher levels, such as the description of a resource (i.e. all triples having the same subject) or instances of a certain class.

Based upon our Semantic Wiki definition and the work with our prototype 3ba.se [6] we identified the following requirements for access control on the underlying RDF triple store:

- Efficiency and scalability should have precedence over expressive power. In modern web applications with complex and dynamic user frontends, query processing has to be as fast as possible. This requirement is more important than expressive power of the access control language since there are usually hundreds of queries to the store triggered by a single web request.
- As a minimal requirement we need context- and content-sensitive triple filtering in a declarative way. This means the access to a tiple set depends on the accounts metadata (e.g. membership information) as well as on the content of the wanted triple set itself (e.g. enforce to give all needed attributes to some resource).
- Access control declaration should be able to use organisational information like command structure and group membership information from inside the controlled or another RDF model. For the most common architectures used for the storage of organisational data inside a company etc., methods exist to migrate or RDF-ify such organizational memory (e.g. [7] for relational

[1] It is important to distinguish between an idealistic wiki with absolutely no access control and realistic wikis, where access control and wiki are not mutually exclusive.

databases and [8] for LDAP directories). It should be possible to use this data to express access control declarations.

In the next section we survey shortly existing research projects which are related to the topic of access control on RDF triple stores.

2 Related work

The research field policy management for the Semantic Web addresses machine interpretable policies to control programs, services and agents on the Web. It is not restricted to security and privacy but also tackles problems related to trust (e.g. trust in resource quality or agents), information filtering, accountability and others. An overview of the current projects is available in the workshop proceedings [9] and [10]).

However, most projects have a different intentions than this work. A policy-based management framework in the sense of [11] aims at an open semantic network environment. In this network the behavior of agents and services is controlled by reasoned decisions over policies. This is necessary due to the complexity of the global approach of controlling all possible agents and services with all possible actions. An example for such a system is Rei [12] which supports specification of policies, analysis and reasoning in pervasive computing applications.

Due to the fact that reasoning procedures are still not scalable to scope with larger knowledge bases, such capabilities can not applied in RDF triple stores today. Another objection to reasoning is the open world assumption, because no external sources are used and access control answers in a closed triple store are limited to yes and no.

Policy management is not only access control but also information filtering based on quality and trust properties. This is necessary whilst operating in a network of distributed resources which are not trustworthy per default. The TriQL.P [13] browser uses queries for filtering information from different sources and qualities. This filter approach is also part of the framework described here.

Another possible approach is the usage of explicit rules (which our approach also makes use of). An example for such a system is [14]. Again, the scope of this system is not an RDF triple store but distributed resources on a network and the access to these resources.

Summarizing we can state that all these systems operate with a different communication model. However, an RDF triple store can be seen as an agent in these frameworks, while the access control layer for the RDF triple store itself operates on a more basic and lightweight model.

3 A Basic Model for Access on RDF Triple Stores

In order that we can develop an access control framework which solves the given requirements, we have to specify a clear communication model for the target environment. In this basic model for access on RDF triple stores, we define three atomic actions:

- *Reading a set of triples* from a stored model: The account queries the triple store with a formal query language (e.g. [15], [16] and [17]) or selects some triples with a more simple method (e.g. a triple pattern). The answer of the triple store is a set of triples which constitutes the intersection of the wanted and the allowed triple set. Most of todays query languages can query not only for submodels. A common result value is a set of variable bindings. Nevertheless there was a requested set of triple which was necessary for the computation of the result set.
- *Adding a set of triples* to a stored model: The account sends the dedicated triple set to the triple store. The store removes all the triples which are not allowed to be written and adds each of the remaining triples, if they are not already contained in the model.
- *Removing a set of triples* from a stored model: The account sends the dedicated triple set to the triple store. The store removes all the triples which are not allowed to be deleted and deletes each of the remaining triples, if they exist in the stored model.

The approach presented in [18] adds more atomic actions here to the above listed ones. They distinguish between one-triple actions, triple-set actions and reasoned-set actions. In this basic model neither one-triple actions nor reasoned-set actions need to be considered because:

- A one-triple action can be seen as a specialization of a triple-set action.
- We define a reasoner application as an agent which holds a specific account with certain rights to a triple store. Following this, reasoned-set actions are combinations of normal triple-set actions which are performed by the reasoner agent.

In the next section, we describe an access control framework which is based on this communication model.

4 Lightweight Framework for Access Control on RDF Triple Stores

As denoted in section 2, we use explicit rules and query filters as the primarily parts of our framework. The whole framework consists of four parts:

- A *query engine* which can apply subset selection query filters to a given model. In this paper we assume that this is a query engine for the SPARQL query language [15] but generally the approach is not limited to a specific query language.
- A *rule processor* which decides whether a query filter is fired for a given action or not. We assume that the used decision rules are described by using the Semantic Web Rule Language (SWRL, [19]) but also, this part can be replaced by an equivalent one.

- A minimalistic *RDF schema* called Lightweight Access Control Schema (LACS, [20]), which describes a basic vocabulary to store rules and query filters.
- The *access control processor*, which starts the query engine and rule processor as needed and maintains some session data.

A fundamental concept of the framework is the presentation of a virtual model to the account instead of the real one. This virtual model is created from the real model and modified through the query filters selected by the rule processor. Thereby, the decision rules can reference to and use resources from the following three different models:

- *Session Model*: This model holds information about the active session (which account is doing what). The triple of this model are dynamically created for every new action on the triple store.
- *User model*: This is the data which the account wants to get access to but it can used by the decision rules too.
- *Maintenance Model*: This model consists of decision rules and filters as well as all other maintenance data like group or account information. The vocabulary for the filter and rule description comes from the lightweight access control schema and from the SWRL specification. The maintenance data which is used by the rules to decide the application of a query filter is not fixed, so rules can be created for every available environment, e.g. a FOAF database or an LDAP backend.

The following example maintenance model consists of two filters and rules. They are created for the following two reading conditions:

- All admins can read every triple.
- All accounts which are from type `foaf:Person`[2] may read only triples where the subject is of type `foaf:Person`.

The rules to effect this behavior are:

$$\begin{aligned}
&\texttt{rdf:type}(\texttt{lacs:CurrentAction}, \texttt{lacs:Read})\\
\wedge\ &\texttt{rdf:sameAs}(\texttt{lacs:CurrentAccount}, ?a)\\
\wedge\ &\texttt{foaf:member}(\texttt{:Admins}, ?a)\\
\rightarrow\ &\texttt{lacs:addAndStop}(\texttt{:currentAction}, \texttt{:AllFilter})
\end{aligned}$$

$$\begin{aligned}
&\texttt{rdf:type}(\texttt{lacs:CurrentAction}, \texttt{lacs:Read})\\
\wedge\ &\texttt{rdf:sameAs}(\texttt{lacs:CurrentAccount}, ?a)\\
\wedge\ &\texttt{rdf:type}(\texttt{foaf:Person}, ?a)\\
\rightarrow\ &\texttt{lacs:add}(\texttt{:currentAction}, \texttt{:FoafOnlyFilter})
\end{aligned}$$

They reference triples in the maintenance model, which describe a group and a member of this groups with the commonly used FOAF vocabulary:

[2] We assume, that the namespaces `rdf`, `rdfs`, `foaf` and `ruleml` are predefined for all examples. The namespace `lacs` is used for the vocabulary described in [20]. All RDF examples are given in Notation 3 (N3, [21]).

```
:Admins a foaf:Group;
  foaf:member :UserSD.

:UserSD a foaf:Person;
  foaf:name "Sebastian Dietzold".
```

These rules reference two query filters. These query filters are given in a specific query syntax and are represented in the RDF with the LACS vocabulary:

```
:AllFilter a lacs:Filter;
  rdfs:label "no restriction filter";
  lacs:sparql "CONSTRUCT { ?s ?p ?o } WHERE { ?s ?p ?o }".

:FoafOnlyFilter a lacs:Filter;
  rdfs:label "read only FOAF address book";
  lacs:sparql """CONSTRUCT { ?s ?p ?o }
    WHERE {?s rdf:type foaf:Person . ?s ?p ?o }""".
```

To give explicit instructions for the access control processor, the rules are represented in RDF and enriched with metadata. The first one is annotated by using the SWRL vocabulary referenced by the namespace `ruleml`, the latter by using the LACS vocabulary.

The next part defines two `lacs:rule` entities which references to SWRL implication rules (not displayed here). Important for the access control processor is the priority of the rules, since the rule selection (see figure 1) is ordered by this property.

```
_:123 a lacs:Rule;
  rdfs:label "Admins can read everything";
  lacs:priority 10;
  lacs:swrlImp [
    a ruleml:imp;
    # ... rule definition ...
    ].

_:321 a lacs:Rule;
  rdfs:label "User can read only foaf:Persons";
  lacs:priority 100;
  lacs:swrlImp [
    a ruleml:imp;
    # ... rule definition ...
    ].
```

Based on this example maintenance model, a sample reading action is processed according figure 1.

First of all, the Session Model is modified by the access control processor to represent the current session. Again, the LACS vocabulary is used:

```
lacs:currentUser = :User2.
lacs:currentAction a lacs:Read.
```

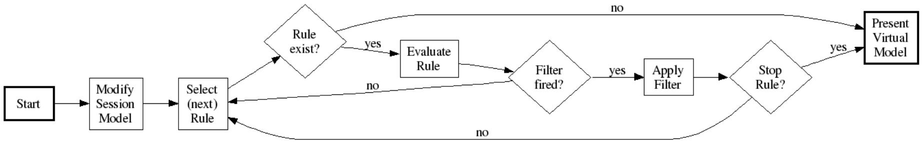


Fig. 1. Rule Processing Flowchart

After that, the first rule is selected and evaluated by the rule engine. In the example maintenance model, this is the "Admins can read everything" rule. Due to this rule, no filter is fired because the user is not member of the admin group. The next rule will be selected and due to this rule, the filter "read only FOAF address book" is fired. So the query process creates the virtual model as it applies filters to the user model. Because there is no other rule, the processor leaves this cycle and presents the virtual model to the user.

This was an example for a reading action. For this type of action, the user query is processed against the virtual model. For writing actions, the filters are not processed against the user model but rather against the model which is supplied by the user (i.e. the triples, he wants to add or delete). After modifying this model according the rules, the add or delete action is processed.

5 Conclusion

We have presented a lightweight access control framework for RDF triple stores based on requirements derived from usage scenarios within a Semantic Wiki application. The basic idea of this framework is the presentation of a virtual model instead of the real one. This model is generated by filtering the original model. Filter are selected by rules. In the examples, we use SPARQL for filtering and SWRL as rule language.

The presented framework strongly depends on the lightweight communication model given in section 3. So it is not intended to be a general access control framework for the Semantic Web. Instead it is designed to be a fast, reliable and easy to implement as part of an RDF triple store. In order to achieve this, we focused on a clear execution algorithm with explicit rules, but do not use any reasoning capabilities.

One important advantage when compared to other approaches is the possibility to create both simple and complex access control environments as necessary. Also, the minimal requirements to the underlying maintenance model are small, so that administrators can maximally reuse existing models within their rules.

References

1. Wilkinson, K., Sayers, C., Kuno, H., Reynolds, D.: Efficient RDF Storage and Retrieval in Jena2. In: Proceedings of First International Workshop on Semantic

Web and Databases 2003. (2003) 131–150

2. Bizer, C.: RAP (RDF API for PHP). Website (2004) http://www.wiwiss.fu-berlin.de/suhl/bizer/rdfapi/.
3. Beckett, D.: The design and implementation of the Redland RDF application framework. Computer Networks **39** (2002) 577–588
4. Broekstra, J., Kampman, A., van Harmelen, F.: Sesame: A Generic Architecture for Storing and Querying RDF and RDF Schema. In Horrocks, I., Hendler, J., eds.: The Semantic Web - ISWC 2002. First International Semantic Web Conference, Sardinia, Italy, June 9-12, 2002, Proceedings. Volume 2342 of Lecture Notes in Computer Science., Springer (2002) 54–68
5. Leuf, B., Cunningham, W.: The Wiki Way. Addison-Wesley Longman, Amsterdam (2001)
6. Auer, S., Dietzold, S., Riechert, T.: 3ba.se Semantic Wiki. Prototype (2006) http://3ba.se.
7. Bizer, C., Seaborne, A.: D2RQ -Treating Non-RDF Databases as Virtual RDF Graphs. Poster (2004) 3rd International Semantic Web Conference (ISWC2004), Hiroshima, Japan.
8. Dietzold, S.: Generating RDF Models from LDAP Directories. In Auer, S., Bizer, C., Miller, L., eds.: Proceedings of the SFSW 05 Workshop on Scripting for the Semantic Web , Hersonissos, Crete, Greece, May 30, 2005. Volume 135 of CEUR Workshop Proceedings., CEUR-WS (2005)
9. Kagal, L., Finin, T., Hendler, J., eds.: Policy Management for the Web. (2005)
10. Kagal, L., Finin, T., Hendler, J., eds.: Proceedings of the Semantic Web and Policy Workshop, held in conjunction with the 4th International Semantic Web Conference, 7 November, 2005, Galway Ireland. (2005)
11. Tonti, G., Bradshaw, J.M., Jeffers, R., Montanari, R., Suri, N., Uszok, A.: Semantic Web Languages for Policy Representation and Reasoning: A Comparison of KAoS, Rei, and Ponder. In Fensel, D., Sycara, K.P., Mylopoulos, J., eds.: The Semantic Web - ISWC 2003, Second International Semantic Web Conference, Sanibel Island, FL, USA, October 20-23, 2003, Proceedings. Volume 2870 of Lecture Notes in Computer Science., Springer (2003) 419–437
12. Kagal, L.: Rei: A Policy Language for the Me-Centric Project. Technical report, HP Labs (2002)
13. Bizer, C., Cyganiak, R., Gauss, T., Maresch, O.: The TriQL.P Browser: Filtering Information using Context-, Content- and Rating-Based Trust Policies. In Kagal, L., Finin, T., Hendler, J., eds.: Proceedings of the Semantic Web and Policy Workshop, held in conjunction with the 4th International Semantic Web Conference, 7 November, 2005, Galway Ireland. (2005) 12–20
14. Li, H., Zhang, X., Wu, H., Qu, Y.: Design and Application of Rule Based Access Control Policies. In Kagal, L., Finin, T., Hendler, J., eds.: Proceedings of the Semantic Web and Policy Workshop, held in conjunction with the 4th International Semantic Web Conference, 7 November, 2005, Galway Ireland. (2005) 34–41
15. Prud'hommeaux, E., Seaborne, A.: SPARQL Query Language for RDF (Working Draft). W3c working draft, World Wide Web Consortium (W3C) (2006)
16. Seaborne, A.: RDQL - A Query Language for RDF. W3c member submission, World Wide Web Consortium (W3C) (2004) http://www.w3.org/Submission/2004/SUBM-RDQL-20040109/.
17. Karvounarakis, G., Alexaki, S., Christophides, V., Plexousakis, D., Scholl, M.: RQL: A Declarative Query Language for RDF. In: Proceedings of the eleventh international conference on World Wide Web, ACM Press (2002) 592–603

18. Reddivari, P., Finin, T., Joshi, A.: Policy based access control for an RDF store. In Kagal, L., Finin, T., Hendler, J., eds.: Policy Management for the Web. (2005) 78–81
19. Horrocks, I., Patel-Schneider, P.F., Boley, H., Tabet, S., Grosof, B., Dean, M.: SWRL: A Semantic Web Rule Language Combining OWL and RuleML. W3c member submission, World Wide Web Consortium (W3C) (2004) http://www.w3.org/Submission/2004/SUBM-SWRL-20040521/.
20. Dietzold, S.: LACS: Lightweight Access Control Schema. OWL ontology (2006) http://purl.org/net/lacs.
21. Berners-Lee, T.: Notation 3 - An readable language for data on the Web. Website (1998) http://www.w3.org/DesignIssues/Notation3.html.

RDFHomepage
or
"Finally, a use for your FOAF file"

Gunnar AAstrand Grimnes, Sven Schwarz, and Leo Sauermann

Knowledge Management
DFKI GmbH
Kaiserslautern, Germany
http://www.dfki.uni-kl.de/∼{grimnes,schwarz,sauermann}
{grimnes,schwarz,sauermann}@dfki.uni-kl.de

Abstract. This paper presents the RDFHomepage project, a framework for using a person's structured data sources to auto-generate an HTML homepage. RDFHomepage uses RDF files as input, and currently supports several well-known RDF schemas, such as FOAF. In addition to these we have RDF converters for other structured file-formats, like Bibtex. RDFHomepage produces valid HTML 4.01 Transitional pages, and makes it easy to roll-out functional homepages for a group of people. The generated HTML code is very general, allowing quick and easy page-redesigning using CSS. RDFHomepage is written in PHP and uses our system for generating PHP classes based on RDF class definitions, enabling quick and easy development of RDF handling PHP code.

1 Introduction

RDFHomepage is a tool for automatic generation of HTML homepages based on RDF files and other structured information sources which a user might already create and maintain. Figure 1 shows an overview screenshot of the default homepage created based on a user's RDF data. This page contains all things one expects on a typical homepage: the top shows the person's name, email, telephone number etc., this is taken from his FOAF profile; further down there is a short bibliography, this taken from a file using the *homepage-schema* we created for this task; the next section lists the projects this user is involved in, taken from the DFKI Organisational Repository (OrgRep); finally the page has a list of people he knows, based on the friends he specified in his FOAF profile. RDFHomepage has the following attractive features:

- Generated pages are valid HTML 4.01 Transitional.
- Generates well structured HTML code that can easily be styled with Cascading Stylesheets (CSS).
- Enables complete separation of content and appearance.
- An RDF Template engine for generating PHP classes for each RDF Schema is used to make the PHP code easy to write and maintain.

- Enables easy rollout of many identical websites across an organisation.

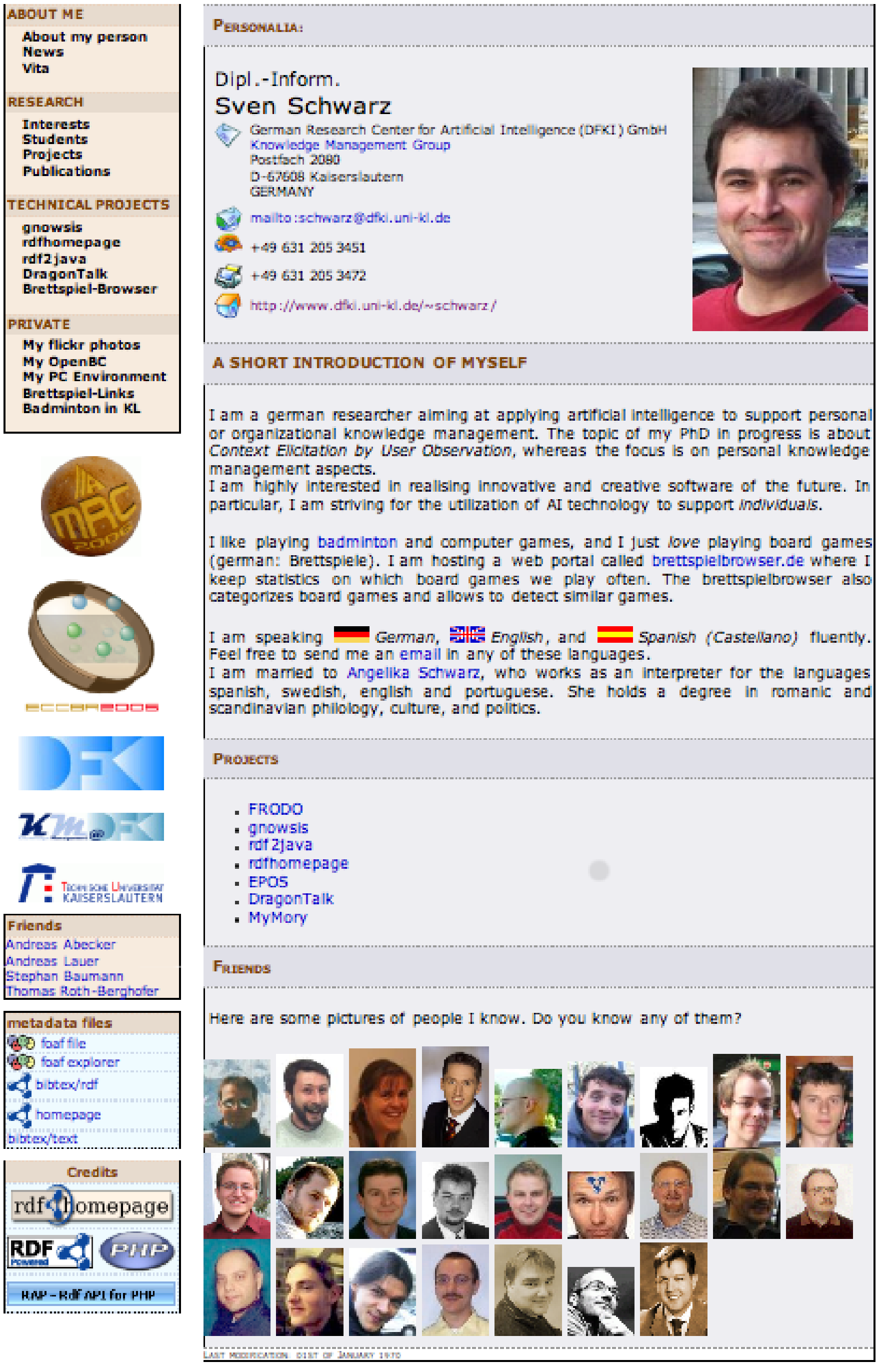
Section 2 outlines the architecture of RDFHomepage. Section 3 discussed the different datasources used by RDFHomepage and Section 4 presents our template engine for generating PHP Classes based on RDF Schemas. Section 5 outlines some future plans and makes concluding remarks.

2 Architecture

For RDFHomepage we chose to use the web-scripting language PHP. We chose PHP because it is free and open-source, and is a powerful and feature complete language, with good support for RDF through the RDF API for PHP (RAP) [1]. PHP is also very often provided by cheap hosting providers, unlike more heavyweight solutions such as Java, making the potential userbase of RDFHomepage much larger. In addition, PHP was also initially known as the *Personal Home Page tools*, (although later renamed to *PHP Hypertext Preprocessor*) and we feel RDFHomepage now brings PHP back to its roots. Alternative techniques that were evaluated include: Treehugger by Damian Steer[2] and Masahide Kanzaki's various XSLT tools[3]. These XSLT tools were not chosen because of the steep learning curve, the (often) silent failures which makes debugging very hard in comparison to PHP scripts. RDFHomepage uses RDF data from several standard sources, detailed in the next section, and in addition to these we created a homepage schema, providing the semantic glue between the other sources, and allowing the user to specify additional personal details in a structured form, for example his interests or personal views on projects. Having explicit representations of these items, rather than HTML pages, enables machine processing of the content, such as automatic aggregation of colleagues with similar interest.

The RDFHomepage distribution comes with a standard set of pages, including "About me", "Projects", "Interests" and "Publications". The side-bar menu can be easily customised to include other static or dynamic pages, enabling RDFHomepage to integrate well with exisiting homepage components. There is also a selection of side-bar boxes available, showing elements like links to the raw RDF sources of the page, or links to companies, projects, etc.

Generation of each page compromising the RDFHomepage can take several seconds, depending on the size of the user's data and of course on the web-server being used. A caching mechanism is therefore a part of RDFHomepage, which will only regenerate pages if the underlying RDF data has changed. Unfortunately normal HTTP/1.1 caching mechanisms could not be used for this, as a single page might be dependent on a number of RDF files, both local and remote, so the caching layer was implemented as an additional PHP layer around the page-generating code.



ABOUT ME
About my person
News
Vita

RESEARCH
Interests
Students
Projects
Publications

TECHNICAL PROJECTS
gnowsis
rdfhomepage
rdf2java
DragonTalk
Brettspiel-Browser

PRIVATE
My flickr photos
My OpenBC
My PC Environment
Brettspiel-Links
Badminton in KL

Friends
Andreas Abecker
Andreas Lauer
Stephan Baumann
Thomas Roth-Berghofer

metadata files
foaf file
foaf explorer
bibtex/rdf
homepage
bibtex/text

Credits

PERSONALIA:

Dipl.-Inform.
Sven Schwarz

German Research Center for Artificial Intelligence (DFKI) GmbH
Knowledge Management Group
Postfach 2080
D-67608 Kaiserslautern
GERMANY

mailto:schwarz@dfki.uni-kl.de
+49 631 205 3451
+49 631 205 3472
http://www.dfki.uni-kl.de/~schwarz/

A SHORT INTRODUCTION OF MYSELF

I am a german researcher aiming at applying artificial intelligence to support personal or organizational knowledge management. The topic of my PhD in progress is about *Context Elicitation by User Observation*, whereas the focus is on personal knowledge management aspects.
I am highly interested in realising innovative and creative software of the future. In particular, I am striving for the utilization of AI technology to support *individuals*.

I like playing badminton and computer games, and I just *love* playing board games (german: Brettspiele). I am hosting a web portal called brettspielbrowser.de where I keep statistics on which board games we play often. The brettspielbrowser also categorizes board games and allows to detect similar games.

I am speaking *German*, *English*, and *Spanish (Castellano)* fluently. Feel free to send me an email in any of these languages.
I am married to Angelika Schwarz, who works as an interpreter for the languages spanish, swedish, english and portuguese. She holds a degree in romanic and scandinavian philology, culture, and politics.

PROJECTS

- FRODO
- gnowsis
- rdf2java
- rdfhomepage
- EPOS
- DragonTalk
- MyMory

FRIENDS

Here are some pictures of people I know. Do you know any of them?

LAST MODIFICATION: 01ST OF JANUARY 1970

Fig. 1. Overview Screenshot of an RDFHomepage Installation

3 RDF Data

3.1 FOAF

The Friend-of-a-Friend ontology [4] was the main point of inspiration for RDFHomepage. A huge number of people in the semantic web community have created their own FOAF profile and published it[1], and there are millions more generated by LiveJournal[2], Ecademy[3] and other social sites producing FOAF. However, there are very few consumers of FOAF files, there have been a range of visualisers and explorers[5], and some proof-of-concepts on using FOAF for distributed authentication[4] and trust [6], but no application that really makes it worth your while to create a FOAF profile. In RDFHomepage the FOAF data is used to create the personalia "About me" page, as well as links to people known by the user, and links to the co-authors of papers on the "Publications" page. Extending and enriching your FOAF file now makes sense: A link to a colleague is missing, just add him to your *foaf.rdf* and all parts of your page are automatically updated.

3.2 Bibtex

BibTeX is a format for managing citations when using TeX or LaTeX. BibTex defines different classes of publications, such as articles, books, theses, etc., and associated optional and required properties of these. Most computer scientists will keep a BibTeX file of their own publications up to date, for use when self-citing or when publishing their papers on their website. There are many tools for helping this process (for example JabRef[5] and BibDesk[6]). Since people already maintain this information in a structured format it makes sense to reuse this information, and to this end we used BibTeX2RDF, written by Wolf Siberski[7]. A sample BibTeX entry converted to RDF is shown as N3 in Figure 2. The output uses largely standard schemas, such as Dublin Core and VCARD, but does also declare its own namespace for BibTeX specific items. (BibTeX2RDF is configurable to output RDF confirming to any ontology, and unfortunately the actual schema for our version not available). As can be seen from the example the authors of a paper are represented as RDF instances of a Person class. Each person gets a URI generated from their name and the ID of a paper they authored, this ensures there are no collisions. BibTeX2RDF will also merge people with exactly the same name (i.e. *smushing*), so there will only be one node for each person. RDFHomepage uses the BibTeX information to create the "Publications" page, merging the BibTeX person nodes with FOAF people, and uses the info from the FOAF file to generate links to co-author's webpages.

[1] http://rdfweb.org/topic/FOAFBulletinBoard

[2] http://www.livejournal.com

[3] http://www.ecademy.com/

[4] http://rdflib.net/doc/user_managenemt/

[5] http://jabref.sourceforge.net/

[6] http://bibdesk.sourceforge.net/

This means users have an interest in keeping the names aligned, and may edit their BibTeX accordingly. We note however, that this is not really a technically satisfying solution. In addition to the author links, RDFHomepage will also link to the PDFs of any papers available for download, as well as generating links to the homepages of the venues where papers are published. The PDF downloads links are taken from BibTeX directly, URL being one of the optional fields of all BibTeX types, and most BibTeX tools make is easy to attach PDFs to citations this way. These links are retrieved from an internal DFKI wiki page listing conference abbreviations and corresponding web-links. To generate links to the corresponding conferences, a social approach was taken. This gathering of relevant conferences and their URLs was already being done in our research-group and the existing Wiki page is now simply parsed looking for HTML links tags with abbreviations as link texts. The conference names and URLs are extracted and matched to the names user in the BibTeX data. This motivated people to keep the Wiki page updated, having a similar effect as seen with the FOAF files.

```
@prefix : <http://www.w3.org/2001/vcard-rdf/3.0#> .
@prefix bibtex: <http://www.edutella.org/bibtex#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .

<aberer2003chatty:ACM_Press> a bibtex:Organization;
  :ADR [
    :Country “USA”;
    :Locality “New York” ];
  :FN “ACM Press” .

<aberer2003chatty:Aberer_Karl> a bibtex:Person;
  :FN “Karl Aberer”;
  :N [
    :Family “Aberer”;
    :Given “Karl” ] .

(...)

<aberer2003chatty> a bibtex:InProceedings;
  dc:creator [
    a rdf:Seq;
    rdf:_1 <aberer2003chatty:Aberer_Karl>;
    rdf:_2 <aberer2003chatty:Cudre-Mauroux_Philippe>;
    rdf:_3 <aberer2003chatty:Hauswirth_Manfred> ];
  dc:date “2003-05”;
  dc:identifier “http://www2003.org/cdrom/papers/refereed/p471/471-aberer.html”;
  dc:publisher <aberer2003chatty:ACM_Press>;
  dc:title “The Chatty Web: Emergent Semantics Through Gossiping”;
  dcterms:isPartOf [
    a bibtex:Proceedings;
    dc:date “2003-05”;
    dc:publisher <aberer2003chatty:ACM_Press>;
    dc:title “Proceedings of the 12th Intl. WWW Conference” ;
    :ADR [
      :Country “Hungary”;
      :Locality “Budapest” ] ];
  bibtex:pages “197-206” .
```

Fig. 2. A BibTeX entry as converted to RDF.

3.3 Projects

At the DFKI information about all projects and the people working in them is centrally maintained in a Organisational Repository (OrgRep)[7] [8]. OrgRep was originally created for use in the FRODO project, but has been maintained and used in many DFKI projects since, for example EPOS, SmartFlow, and MyMory. The organisation ontology is used in EPOS to infer relevant contextual information, i.e. given a person as input the system can look up relevant projects, and vice versa. An example project entry from OrgRep is shown in Figure 3. In addition to the fields shown here OrgRep also contains a longer general description of a project as HTML data, this is used on the "Project" page, unless the user choses to override this with his own personal view on the project. Again the heavy reuse of the data in several contexts increased the motivation to keep your data up-to-date.

```
@prefix : <http://km.dfki.de/model/org#> .
@prefix a: <http://dfki.rdf.util.rdf2java/default#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .

:rdfhomepage a :Project;
  :belongsTo :OrganisationalModel_00095;
  :containsMember a:id_20040921_120247_156f920,
      :GuentherNoack,
      :MalteKiesel,
      :OrganisationalModel_00077,
      :OrganisationalModel_00102;
  :homepage "http://rdfhomepage.opendfki.de/";
  :logo "http://www.dfki.uni-kl.de/~schwarz/rdfhomepage/images/rdfhomepage06-180.png";
  :managedBy :OrganisationalModel_00077;
  :name "rdfhomepage";
  rdfs:label "rdfhomepage" .
```

Fig. 3. Example OrgRep Project Entry.

4 RDF Class Templates

Working with RDF on the triple level can be very difficult and time consuming for developers and it introduces lots of scope for errors. To facilitate this process we created an RDF Template Generator for PHP, this takes one or more RDF Schema documents and produces a PHP class definition based on a selected RDF Class. This means that programmers no longer have to deal directly with the RDF API, but can continue work on the level of PHP objects which they are more used to. An example PHP Class outline generated from the OrgRep ontology is shown in Figure 4. Note how the properties of the OrgRep have

[7] The OrgRep ontology can be downloaded from http://www.dfki.uni-kl.de/~grimnes/2006/03/orgrep/orgrep.rdfs

been mapped to getter-methods of the form *get_namespace_localname*, where the namespace is abbreviated. Each getter method takes a boolean argument specifying whether an array or a single value should be returned. Arrays are useful when a property is either multi-valued or the value is an RDF list or collection, if multiple values are present but only a single value requested a random one is returned. The values returned from these getters are either instances of other generated template classes, depending on the value being a typed resource and the appropriate class existing, or the raw RDF nodes otherwise. Note also that since RDFHomepage only does processing of RDF data, there are no setter-methods generated, however, these could easily be added following the same pattern. The template generator allows loading of multiple schemas before creating the classes, as is illustrated by the *rdfhome_htmltext* property shown in the example, which originates from the homepage schema, but still has an *rdf:domain* of an *OrgRep project*. The template generator also supports inference, so for example: in the case of FOAF, one can generate a *Person* class which includes the properties from *Agent* (a super class of *Person* in the FOAF schema). The generates templates build on top of RAP and can seamlessly fit into a project already using the RAP API.

The approach of mapping RDF objects to a the Object Orientation framework of a programming language is not new and is similar to many other projects, for example: Tramp [9] by Aaron Swartz (the original as far as we are aware) did this for Python; Sparta [10], a more up-to-date Python solution; ActiveRDF [11], a Ruby version; and several Java variants, for example RDFReactor [12] and rdf2java [13]. Outside the RDF community this approach was also used for mapping relational databases to objects, as done with for example Enterprise Java Beans, or the SQLObject in Python[8]. There also exists a library that provides this functionality for PHP, called RDF World [14], but it is different from our approach in that all the RDF processing is done at runtime, whereas we pregenerate the class-templates. This means our solution is much faster when running, and also removes the need to have the triples of the RDF schemas in memory during runtime.

4.1 Evaluation

As shown by the large number of projects offering different solutions for mapping RDF data to classes in object-orientation frameworks, this is a very natural idea. As we see it there are two main reasons to use this approach:

1. Hiding the complexities of RDF – most programmers are comfortable with thinking in terms of objects and classes, whereas working with RDF graphs can be very complicated.
2. Enabling code-autocompletion – modern integrated development environments can be a tremendous help when writing object oriented code, and removes the need to know the ins and outs of the APIs one is using. With

[8] http://www.sqlobject.org/

```
<?
class org_Project {
  function org_Project($uri,$model)

  function get_rdfhomepage_htmlText($a=false)
  function get_org_eMail($a=false)
  function get_org_homepage($a=false)
  function get_org_projectFlyer($a=false)
  function get_org_startYear($a=false)
  function get_org_endYear($a=false)
  function get_org_logo($a=false)
  function get_org_containsMember($a=false)
  function get_org_hasProfile($a=false)
  function get_org_informalDescription($a=false)
  function get_org_managedBy($a=false)
  function get_org_name($a=false)

  function getProperty($s,$p,$m=False)
}
?>
```

Fig. 4. Example of Generated PHP Class Outline.

pre-generated code for RDF classes this convenience can also be used with RDF.

3. Adding custom code to wrapper – by creating classes that inherit from the generated classes one can embed functionality specific to particular classes in the most natural place. For example, *foaf:Person* could have an added function for automatically generating sha1 checksums of the email address.

With regard to the first point our RDF Templates were only partly successful, the users still have to understand how multiple values for a single property work in RDF, and still have to understand the difference between resources, literals and anonymous nodes. With regard to the second point, full code-completion is possible with our solution, and was used by several of the coders during devlopment. Code-completion being possible is another positive effect of pre-generating the classes, a downside of this pre-generation is that we are tied to a static schema, and cannot directly deal with properties that do not conform. For instance, if an instance of *foaf:Person* has a *middleName* property (which doesn't appear in the FOAF specification), it cannot be accessed using a get method. However, a general *getProperty* method is generated to work around this.

5 Conclusion & Future Work

RDFHomepage provides a quick and easy to way to create attractive and informative homepages, with all the content typically found on a research active user's homepage. The pages are generated from RDF files allowing complete separation of content and appearance, as well as making the homepage machine processable by semantic web agents. Editing of a few simple configuration files makes customising the generated pages trivial, and if the customisation offered is not sufficient the use of auto-generated PHP classes makes writing additional PHP code to handle RDF very easy. The scenario of "creating a homepage" proved to be a good playground for Semantic Web technologies and as the commercial success of facebook.com and myspace.com show, it is a very active area. Deploying RDFHomepage at our department triggered users to keep their FOAF files, BibTeX files, the organisational repository and internal Wiki pages updated, and this in turn leads to more accurate date in other projects. We have several plans for further developments of RDFHomepage, primarily we continue to add sources of structured information that user's may want to add to their homepage. Things we planning to add very soon include:

- Flickr[9] photo streams – for example, a side-bar box showing thumbnails of the user's last photos.
- RSS/Atom support – listing the last entries in the user's blog.
- Calendar support – where is the user today? What are his plans this week? This could for instance make use of the RDF Calendaring efforts[10], which include converters between iCal and RDF.

There is also scope for improving our template system: to make it a more general solution for RDF based software engineering we need to generate "setters" as well as "getters" for the properties. Another important addition is to add support for data-types that exist in PHP, and automatically convert fields that represent for example dates to a standard format.

RDFHomepage is open source, release under a GNU Public License (GPL) and can be downloaded from http://rdfhomepage.opendfki.de. To get a better impression of what RDFHomepage can do, consider visiting any of the author's homepages, they are all automatically-generated![11]

References

1. Radoslaw Oldakowski, Christian Bizer, D.W.: RAP: RDF API for PHP. In: Workshop on Scripting for the Semantic Web (SFSW 2005) at 2nd European Semantic Web Conference (ESWC 2005). (2005)
2. Steer, D.: TreeHugger. (http://rdfweb.org/people/damian/treehugger/)
3. Kanzaki, M.: XSLT Tools. (http://www.kanzaki.com/info/who.html.en)
4. Brickley, D., Miller, L.: FOAF Vocabulary Specification (2006) http://xmlns.com/foaf/0.1/.
5. Fredriksen, M.: FOAF Explorer. (http://xml.mfd-consult.dk/foaf/explorer/)
6. Golbeck, J., Parsia, B., Hendler, J.: Trust Networks on the Semantic Web. In: Seventh International Workshop on Cooperative Information Agents (CIA-03), Helsinki, Finland (2003) 238–249
7. Siberski, W.: BibTeX2RDF. (http://www.l3s.de/~siberski/bibtex2rdf/)

[9] http://flickr.com

[10] http://www.w3.org/2002/12/cal/

[11] See also http://rdfhomepage.opendfki.de/cgi-bin/trac.cgi/wiki/WorkingInstallations

8. van Elst, L., Abecker, A., Bernardi, A., Lauer, A., Maus, H., Schwarz, S.: An Agent-based Framework for Distributed Organizational Memories. In Bichler, M., Holtmann, C., Kirn, S., Mller, J.P., Weinhardt, C., eds.: Coordination and Agent Technology in Value Networks, Multikonferenz Wirtschaftsinformatik (MKWI-2004), 9.-11.3.2004, Essen, GITO-Verlag, Berlin (2004) 181–196
9. Swartz, A.: (TRAMP: Makes RDF look like Python data structures.) http://www.aaronsw.com/2002/tramp.
10. Nottingham, M.: (Sparta: a Simple API for RDF) http://www.mnot.net/sw/sparta/.
11. Oren, E.: (ActiveRDF - putting the semantic web on rails) http://activerdf.m3pe.org/.
12. Völkel, M., Sure, Y.: RDFReactor – From Ontologies to Programmatic Data Access. In: Poster Proceedings of the Fourth International Semantic Web Conference. (2005) http://rdfreactor.ontoware.org/.
13. Sven Schwarz, M.K., Sintek, M.: (RDF2Java) http://rdf2java.opendfki.de.
14. Snyder, C.: (Rdfworld.php) http://chxo.com/rdfworld/index.htm.

Brainlets: "instant" Semantic Web applications

Giovanni Tummarello, Christian Morbidoni, Michele Nucci, Onofrio Panzarino

SeMedia group, Dipartimento di Elettronica, Intelligenza Artificiale e Telecomunicazioni,
Università Politecnica delle Marche, via Breccie Bianche, 60100 Ancona
g.tummrello@gmail.com, c.morbidoni@deit.univpm.it, mik.nucci@gmail.com

Abstract
In this paper we present the "Brainlet" paradigm, a way to create rich Semantic Web user interfaces and interaction environments. Brainlets are half way between configuration files and light scripts and are "executed" by the DBin rich Semantic Web Platform. The main motivation behind Brainlets is enabling domain experts, rather than programmers, to create rich Semantic Web environments and communities. Brainlets can in fact be created simply by XML configuration files and XML based scripts along with the proper ontologies. Advanced Brainlets can be created based on the internal DBin API and/or the Eclipse Rich Client platform API. Brainlets are distributed as plug-ins and enable user communities, providing a common 'vision' of the domain and tools, to collectively create and exploit rich Semantic Web datasets.

1. Introduction

A fundamental design goal of the DBin rich Semantic Web platform [1] has been the ability to enable domain experts, rather than Semantic Web programmers and hackers, to independently start user communities about any topic of choice. We obtain this result with the definition of a scripting/configuration framework which enables the creation of rich Semantic Web interaction environments using tools such as XML and OWL editors. Such environments, which an end user might even perceive as a full featured domain specific applications, are named Brainlet. This paper illustrates their creation process and features.

Brainlet are distributed to the users as plug-ins (e.g. posted on a web page), thus enabling communities to share a common vision of the domain and a set of tools to collectively create and exploit rich Semantic Web datasets. In order to understand the need for the features offered by the Brainlet paradigm, lets first address the question: *in which way one can create a 'community' of 'Semantic Web users' ?*

Traditional Internet user aggregation channels such as IRC, newsgroups and web forums make it rather straightforward to create user communities but interactions among users are limited to textual messages. Finding information on such channels is, in the best scenario, only possible trough textual search, structured data is not tractable.

On the other hand, the goal of Semantic Web based user communities - supported by the DBin platform - is to enable the cooperative creation and exploitation of semantically structured knowledge bases. It is clear how a standard message board is a very simple subset of the possibilities enabled by such Semantic Web communities.

In this paper we assume that issues such as metadata exchange algorithm and trust based filtering are, among others, solved by the facilities which the DBin platform includes. These are the RDFGrowth [2] P2P algorithm, providing topic based *metadata exchange channels*, and the RDF Digital Signature methodology [3].

In this condition, Semantic Web communities startup basically boils down to create domain specific user interfaces which shields the users of the communities from complexities and at the same time provides domain specific browsing, editing and querying tools.

Providing a generic semantic web GUI is all but a trivial task; a lot of approaches have been proposed recently, among which [4], [5], [6], [7], [8]. While pro and cons can be argued for each specific solution, it is clear that user interface issues are complex ones and that it will be very difficult to think about a single solution that can be as usable as ad hoc ones for each specific domain. Aware of this, we propose an approach based on the aggregation of simple, generic components coordinated in a domain specific way by a mix of XML/OWL and scripting technologies.

2. DBin and background scenario

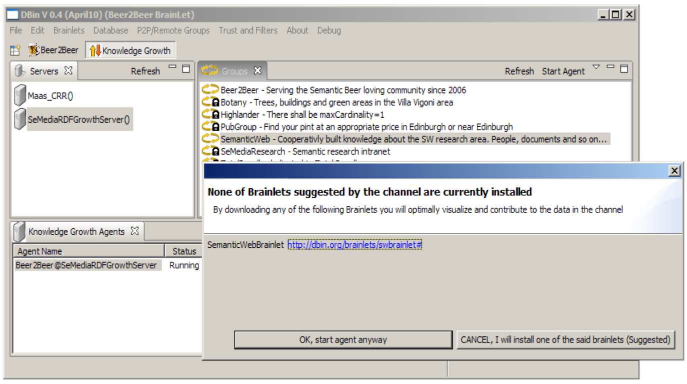
A typical use of DBin might be similar to that of popular file sharing programs, the purpose however being completely different. While usual P2P applications “grow” the local availability of data, DBin grows RDF knowledge. Once a user enters a metadata exchange channel, RDF annotations just start flowing in and out “piece by piece”.

For example, a user who expresses interest in a particular topic (say “Beers” as in DBin demo group) will keep a DBin open, possibly minimized, connected with a related P2P knowledge exchange channel where relevant information bits will be collected from and exchanged with other participants. Such information bits might be pure metadata annotations (e.g. “the alcoholic content of beer X is Y”) but also advanced annotations containing pointers to rich media (e.g. a picture of the glass, a PDF document, a Wikipedia page etc..).

With the support of the proper Brainlet, the user could then browse, reply or further annotate such information bits either for personal use or to contribute to the group knowledge. If such replies include attachment data (e.g. a picture), DBin automatically takes care of the needed web publishing using services such as that offered at *http://public.dbin.org* . At database level, the information is stored in DBin as RDF; At the user level, however, the common operations and views are grouped in domain specific user interfaces, the Brainlets. The installation of a specific Brainlet (e.g. targeted to the Beer domain), is not necessary to connect to the group and receive information but it enables users to visualize and edit such domain specific annotations, which, without proper domain knowledge and settings, are only 'row' triples in the DB.

3. Brainlets: creation, configuration and scripting

Brainlets are plug-ins (technically Eclipse Rich Client Platform [9] compliant plug-ins) that can be installed into the DBin platform simply copying a file in a proper directory. When a user selects a metadata exchange channel the remote server might suggest the use of a specific Brainlet to best experience the information contained in the group and participate in the annotation. Such suggestion is made by whoever created the group, very likely but not necessarily, the same person who created the Brainlet. Figure 1 shows an actual runtime screen-shot.



Maas_CRR()
SeMediaRDFGrowthServer()

Beer2Beer - Serving the Semantic Beer loving community since 2006
Botany - Trees, buildings and green areas in the Villa Vigoni area
Highlander - There shall be maxCardinality=1
PubGroup - Find your pint at an appropriate price in Edinburgh or near Edinburgh
SemanticWeb - Cooperativly built knowledge about the SW research area. People, documents and so on...
SeMediaResearch - Semantic research intranet

Agent Name	Status
Beer2Beer@SeMediaRDFGrowthServer	Running

None of Brainlets suggested by the channel are currently installed

By downloading any of the following Brainlets you will optimally visualize and contribute to the data in the channel

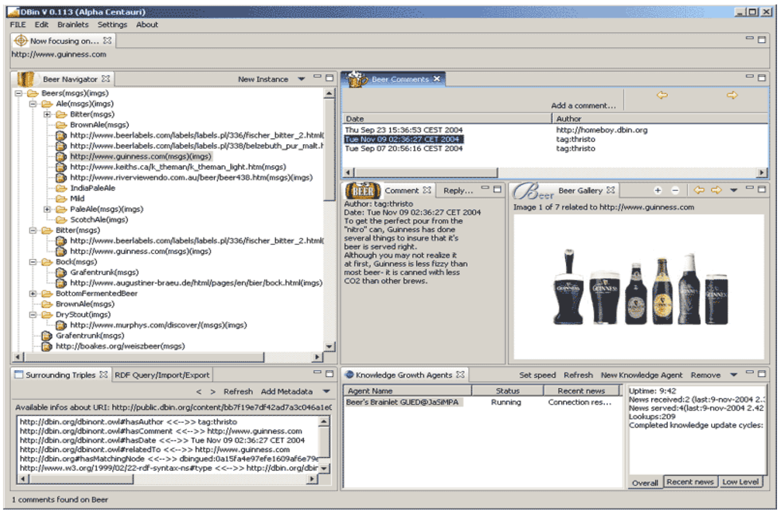
SemanticWebBrainlet http://dbin.org/brainlets/swbrainlet#

Figure 1: Joining a metadata exchange channel (right window) offered by a RDF-Growth server (left window) in the DBin platform. In this case the server suggests the user to download a specific Brainlet to better interact with the knowledge exchanged.

Brainlets are composed of XML, OWL ontologies and scripts. As an overview, they contain:

- The ontologies to be used for annotations in the domain (e.g. The beer ontology);
- A general GUI layout: which components to visualize (e.g. A message board, an ontology browser, a “detail” view) and how they are cascaded in terms of selection/reaction;
- Templates for domain specific “annotations”, e.g. a “Movie” Brainlet might have a “review” template that users fill. This allow a “reviews” view to have useful orderings based on the known fields in the review. The GUI for the templates is generated automatically;
- Templates for readily available, “pre-cooked” domain queries, which are structurally complex domain queries with only a few simple free parameters, e.g. “give me the name of the cinema where the best movie of genre X is being shown tonight”;
- A suggested trust model and information filtering rules for the domain. e.g. public keys of well known “founding members” or authorities, preset “browsing levels”;
- A script for guiding the user in creating new URIs for domain resources, e.g. inserting a new "beer" in the DB.
- Scripts connected to Brainlet specific menus or buttons and that implement domain specific functions;
- Support material, customized icons, help files etc..
- Supporting Java code and libraries;
- A basic RDF knowledge package, conforming to the information shared in a specific group.

Almost all of these elements are optional in a Brainlet. Figure 2 shows a typical runtime view of the Beer2Beer Brainlet (*http://www.dbin.org/brainlets/beer2beer*). A Semantic Web Research Brainlet, which demonstrates how the DBin platform can naturally handle completely different domains of interest, is available at *http://www.dbin.org/brainlets/swbrainlet*. The following sections are concerned with how the specific features and aspects listed above can be configured.



http://www.guinness.com

```
Beers(msgs)(imgs)
  Ale(msgs)(imgs)
    Bitter(msgs)
    BrownAle(msgs)
    http://www.beerlabels.com/labels/labels.pl/336/fischer_bitter_2.html(
    http://www.beerlabels.com/labels/labels.pl/338/belzebuth_pur_malt.h
    http://www.guinness.com(msgs)(imgs)
    http://www.keiths.ca/k_theman/k_theman_light.htm(msgs)
    http://www.riverviewendo.com.au/beer/beer438.htm(msgs)(imgs)
    IndiaPaleAle
    Mild
    PaleAle(msgs)(imgs)
    ScotchAle(imgs)
  Bitter(msgs)
    http://www.beerlabels.com/labels/labels.pl/336/fischer_bitter_2.html(
    http://www.guinness.com(msgs)(imgs)
  Bock(msgs)
    Grafentrunk(msgs)
    http://www.augustiner-braeu.de/html/pages/en/bier/bock.html(imgs)
  BottomFermentedBeer
  BrownAle(msgs)
  DryStout(imgs)
    http://www.murphys.com/discover/(msgs)(imgs)
  Grafentrunk(msgs)
  http://boakes.org/weiszbeer(msgs)
```

Date	Author
Thu Sep 23 15:36:53 CEST 2004	http://homeboy.dbin.org
Tue Nov 09 02:36:27 CET 2004	tag:thristo
Tue Sep 07 20:56:16 CEST 2004	tag:thristo

Author: tag:thristo
Date: Tue Nov 09 02:36:27 CET 2004
To get the perfect pour from the "nitro" can, Guinness has done several things to insure that it's beer is served right.
Although you may not realize it at first, Guinness is less fizzy than most beer- it is canned with less CO2 than other brews.

Image 1 of 7 related to http://www.guinness.com

Available infos about URI: http://public.dbin.org/content/bb7f19e7df42ad7a3c046a1e(

```
http://dbin.org/dbinont.owl#hasAuthor <<-->> tag:thristo
http://dbin.org/dbinont.owl#hasComment <<-->> http://www.guinness.com
http://dbin.org/dbinont.owl#hasDate <<-->> Tue Nov 09 02:36:27 CET 2004
http://dbin.org/dbinont.owl#relatedTo <<-->> http://www.guinness.com
http://dbin.org#hasMatchingNode <<-->> dbingued:0a15fa4e97efe1609af6e79
http://www.w3.org/1999/02/22-rdf-syntax-ns#type <<-->> http://dbin.org/dbir
```

Agent Name	Status	Recent news
Beer's Brainlet GUED@JaSiMPA	Running	Connection res...

Uptime: 9:42
News received:2 (last:9-nov-2004 2.:
News served:4(last:9-nov-2004 2.42
Lookups:209
Completed knowledge update cycles:

1 comments found on Beer

Figure 2 A screen shot of the Beer2Beer Brainlet running. The principal "views" are: an ontology (and instances) browsing Navigator, the Knowledge Agents view, showing statistics about the currently running knowledge agents, and a set of "Annotation" views.

3.1. From Eclipse plug-in to Brainlet

To create a Brainlet, it is necessary to fill a given empty template which configures an eclipse plug-in. To append a new "Brainlet" to the list of those known by DBin, one have to add the following XML in the standard eclipse *plugin.xml* file:

```
<extension point="org.dbin.gui.brainlet.core.xmlbrainlet">
   <brainlet perspectiveID="org.dbin.gui.brainlet.beer" file="brainlet/beer.xml"/>
</extension>
```

Other configuration options in the *plugin.xml* will include the invocation and positioning of any desired DBin predefined basic views such as the Navigator (section 3.2), annotations, gallery etc..

3.2. Brainlet configuration in detail

Let's now see the content of the *beer.xml* file, the main Brainlet configuration which has been specified in the previous section as an attribute of the `org.dbin.gui.brainlet.core.xmlbrainlet` entry point.

A Brainlet has a name, a version, a URI identifying it, usually pointing at the Brainlet download location (to be used in the dialog show in Figure 1), and other basic information, specified as attributes of the XML corresponding element. The following is the declaration for the Beer-to-Beer Brainlet:

```
<Brainlet name="Beer2Beer" author="Onofrio Panzarino" version="1.0"
  uri="http://dbin.org/brainlets/beer2beer#" banner="Targeted to beer's fans.
```

```
        Join to learn and share your knowledge about your favorite beers.">
```

Ontologies and base domain knowledge

First step in creating a new Brainlet is the choice of appropriate ontologies to capture the concepts of the target domain. Once existing ontologies have been identified and, if needed, new ones have been developed, the corresponding OWL files are usually included and shipped in the Brainlet itself, although they could be placed on the Web. Each of them will be declared in the XML file, specifying the location of the OWL file, a unique name for the ontology and it's base namespace:

```
<Ontology file="brainlet/ontologies/beer.owl" name="beer"
        base="http://www.purl.org/net/ontology/beer#"/>
<Ontology file="brainlet/ontologies/BeerAnnotations.owl"
        name="beer_annotations" base="http://dbin.org/beer#"/>
```

A Brainlet might need some background RDF data, a starting knowledge base shared by every client using the same Brainlet. The following line cause an RDF file to be installed into DBin:

```
<Data file="brainlet/rdfdata/beer.rdf" base=""/>
```

Tree based navigation of domain resources: the main Navigator

The way concepts and instances are presented and browsed is crucial to the usability of the interface and the effectiveness in finding relevant information. Graph based visualizers are notably problematic when dealing with a relevant number of resources. For this reason, the solution that the main DBin Navigator provides is based on flexible and dynamic tree structures. Such approach can be seen to scale very well with respect to the number of resources, e.g. in Brainlets such as the SW Research one (*http://www.dbin.org/brainlets/swbrainlet/*).

The peculiarity of the approach is that every Brainlet creator can decide which is the 'relation' between each tree item and its children by the use of scripts or semantic web queries (in DBin these are currently expressed in SeRQL syntax [10], but support for other query languages is likely to be provided in future versions). The basic use of such facility is to provide an ontology based navigation. Creating a tree branch by which a class hierarchy based navigation of Beers is as simple as:

```
<Topic name="Beers by Type" uri="http://www.purl.org/net/ontology/beer#Beer">
   <Child query="SELECT X FROM {X} serql:directSubClassOf {$parent}
                WHERE X != $parent">
     <Child subjectBy="serql:directType” icon="/icons/beer.gif"/>
   </Child>
    <Child subjectBy="serql:directType" icon="/icons/beer.gif"/>
</Topic>
```

Each Topic has an associated URI (in this case that of the base class "Beer"), that is every tree conceptually starts from an RDF resource. Topic children are then recursively defined by the results of queries involving the parent resource. In the case of a first level Child, the parent resource will be the resource associated to the entailing Topic. There can be multiple topic branches configured in the Navigator. For example a "Beers by Brewery" branch is configured as follows:

```
<Topic name="Beers by Brewery" uri="http://www.purl.org/.../beer#Region">
   <Child query="SELECT X FROM {X} rdf:type {$parent} WHERE X != $parent">
      <Child subjectBy="http://www.purl.org/net/ontology/beer#brewedBy"/>
   </Child>
</Topic>
```

Note that, of course, different conceptual paths can lead to the very same resource. Figure 3a shows a run time screen-shot of a Brainlet configured with the Navigator Topics that have been just illustrated.

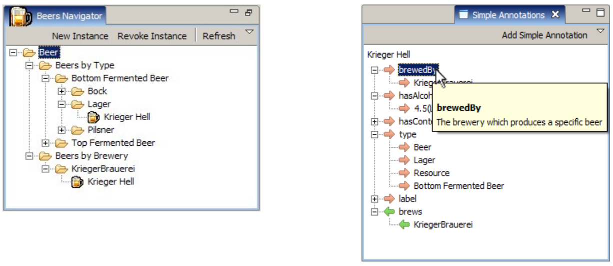


Figure 3: a) The simple annotation view shows the properties of the selected resource (left figure). b) A navigator view generated by the configuration shown above. The instance of a beer can be reached following different paths, by type or by brewery(right figure).

Selection flow across Brainlets parts

At user interface level, as shown in Figure 2, a Brainlet is composed by a set of 'view parts' (as they're called in Eclipse parlor). Usually, each part takes a resource as a main "focus" and shows distinct aspects of the same RDF knowledge 'around' this resource. The Simple Annotations view, for example (Figure 3b), shows the outgoing and incoming RDF statements surrounding the selected RDF node.

As mentioned, the Brainlet creator decides the configurations of the standard view so to have a personalized title, icon, position and so on. Selection flows are also scripted at this point; it is possible to establish the precise cause effect chain by which selecting an icon on a view will cause other views to change. This is a powerful way to customize the flow of information within the user interface. Let's consider the following XML code:

```
<View id="GUEDNavigator" title="Beers Navigator"
        icon="icons/nav.gif" selectorFor="main" listenTo="main" />
<View id="Annotations" listenTo="main" selectorFor="annotation" />
<View id="Content" title="Details"
        listenTo="annotation" selectorFor="annotationContent,main" />
<View id="SimpleAnnotationsView"
        listenTo="main" selectorFor="annotationContent" />
```

Using the attributes 'listenTo' and 'selectorFor', views act both as listeners or as notifiers of "Selection Managers". A view is listener of a selection manager when it will change its content when the manager notifies the change of URI and is a "selector" for a manager when some user action within the view itself will cause the manager to change its selected URI. Many "selection manager" can be created to support the information flow logic within the Brainlet.

Assisted querying tools

Within a specific domain there are often some queries that are frequently used to fulfill relevant use cases. Continuing our "Beer" example, such a query could be “find beers [stronger|lighter] than X degrees”. The "precooked queries" facility gives the Brainlet creators the ability to provide such "fill in the blanks" queries to end users.

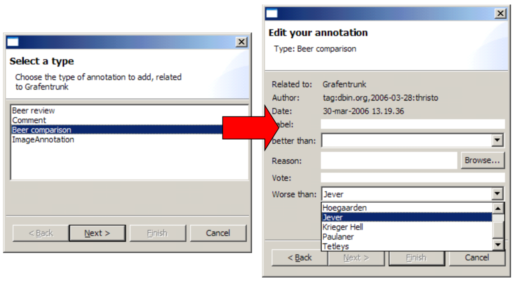
Precooked queries can also be scripts stored in Java classes methods or executed via an interpreter. Here is an example of a Precooked Query:

```
<PrecookedQuery label="Beer Strength"
    message="All beer with alcoholic content greater or lesser than $degree"
    query="SELECT * FROM {X} rdf:type {beer:Beer}; beer:hasAlcoholicStrength {Y}
            WHERE Y $operator $degree" USING NAMESPACES
            beer = http://www.purl.org/net/ontology/beer#
    clipboardQuery="CONSTRUCT {X} rdf:type {dbin:SemClipBoardUri}
                    FROM {X} rdf:type {beer:Beer}; beer:hasAlcoholicStrength {Y}
                    WHERE Y $operator $degree USING NAMESPACS
                              dbin = http://dbin.org/dbinont.owl#,
                              beer = http://www.purl.org/net/ontology/beer#" >
   <Variable name="$operator" legalValues="&lt;,&gt;" defaultValue="&gt;" />
   <Variable name="$degree" />
</PrecookedQuery>
```

In this configuration fragment, a precooked query named "Beer Strength" will show the user a human readable query and allow to chose the operator (< or > , default value >) and a variable value. Such inputs will be used both in a select query, used to find the elements and display them as a table, and in a construct query, used to construct a Semantic Clipboard content.

The Semantic Clipboard is another element that a Brainlet creator might chose to make visible and serves as a temporary graph where the user can put content which can be then passed to other visualizers (e.g. the map viewer as shown below) or external applications by serializing its content to the system clipboard.

Note that a precooked query might link concepts from different domains, and different Brainlets. Suppose to have a Beer Brainlet and a Pub Brainlet. A precooked query as 'Find all the pubs serving beer X which does not contain ingredient Y' is a cross domain query and obviously would only work once sufficient knowledge has been extracted from the two groups.



Select a type

Choose the type of annotation to add, related to Grafentrunk

Edit your annotation

Type: Beer comparison

Related to: Grafentrunk
Author: tag:dbin.org,2006-03-28:thristo
Date: 30-mar-2006 13.19.36
Label:
better than:
Reason:
Vote:
Worse than: Jever

Figure 4: Advanced annotations are defined in OWL and auto generate property visualization and editing interfaces.

Domain specific annotations

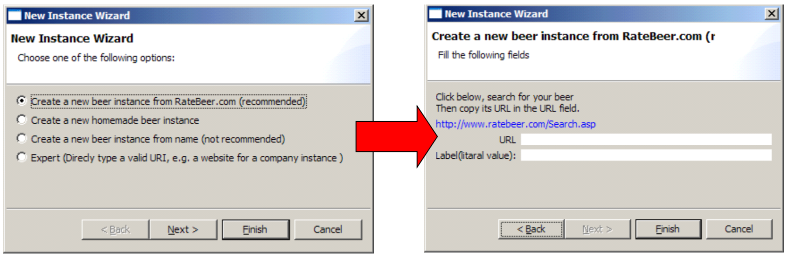
Brainlets assist the users in creating simple annotations, that is in linking resources with properties. Ontology based wizards suggest both the properties that are applicable to the given resource and instances that are currently known to the database and that are of a kind appropriate for a selected property.

A Brainlet creator can however also chose to create "complex annotation types" using a specially defined OWL ontology. An example of such complex annotations is the "Beer Comparison" annotations, which directly compare beers stating which one is better or worse and why. Upon selecting "Add advanced annotation" in DBin the system determines which advanced annotations can be applied to the specified resource and provides a wizard as shown in Figure 4.

Identifier assignment facilities

In our scenario each user is entitled to add new concepts into a knowledge base that is shared within a P2P group. A methodology is needed to make so that different persons, independently adding the same concept (e.g. a particular beer: Peroni Light) into the system, will choose, with a reasonable probability, the same URI to identify it. It is evident that this is necessary for the annotations about Peroni Light to be exchanged among peers, as well as to avoid duplication of concepts. A possible solution might be that of suggesting the user to visit a web site (e.g. RateBeer.com), chosen from the Brainlet creator, where a web page referring to Peroni Light can be found, and to use the relative URL to identify the concept.

Depending on the type of instance one is adding (a beer, a book, a movie...), different methodologies could be thought to create (or find) a proper URI. In the case of a well known concept in a particular domain, for example, we can assume that everybody would reasonably refer to it using the same word, and we can build a new URI with this word as fragment identifier. Note that such procedures does not ensure that different users will end up with the same URI, but still can work in a lot of cases.
Within a Brainlet it is possible to define a set of *URIWizards*, basically scripts that implement a given procedure and guide the user in creating a URI, and to assign them to a class. Once instance of this class is being inserted these scripts are activated providing the user with a graphic wizard, as shown in Figure 5 .



Choose one of the following options:

● Create a new beer instance from RateBeer.com (recommended)
○ Create a new homemade beer instance
○ Create a new beer instance from name (not recommended)
○ Expert (Diredy type a valid URI, e.g. a website for a company instance)

Create a new beer instance from RateBeer.com (r

Fill the following fields

Click below, search for your beer
Then copy its URL in the URL field.
http://www.ratebeer.com/Search.asp
URL
Label(litaral value):

Figure 5: Choosing to insert a new instance of beer the user is prompted with a wizard and can choose different procedures to create a URI.

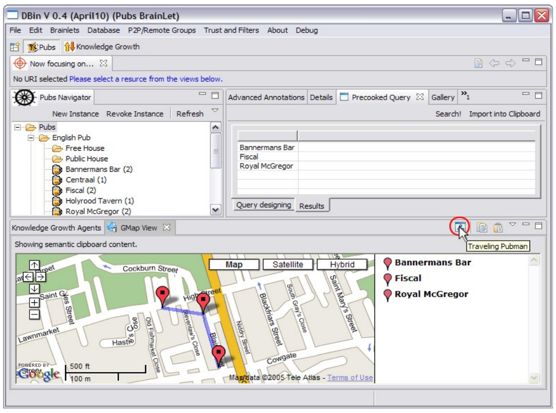
An XML example configuration is given:

```
<Class uri="http://www.purl.org/net/ontology/beer#Beer">
   <FromUrlWizard label="Create a new beer from RateBeer.com (recommended)">
      <Message content="Click below, search for your beer &#10;
                         Then copy its URL in the URL field."/>
      <Website>http://www.ratebeer.com/Search.asp</Website>
      <UrlPrefix>http://www.ratebeer.com/Beer/</UrlPrefix>
   </FromUrlWizard>
   <FromPropertiesWizard label="Create a new homemade beer instance">
        <Property label="Label" prefix="$useruri:HomemadeBeer:"
                uri="http://www.w3.org/2000/01/rdf-schema#label"/>
   </FromPropertiesWizard>
</Class>
```

Domain specific script inclusion

In developing DBin we encountered the need to implement some actions which, in a sense, are outside of the scope of the platform itself but yet are needed in order to provide functionalities for the user to experience the power of underlaying semantic knowledge.

For example we wanted the user to be able to organize a pubs tour, choosing a list of pubs (e.g. using precooked queries to find the pubs that serve a particular beer) and automatically calculating the optimal order of pubs to be visited. This is a very specific task and we didn't want to modify the platform architecture in order to solve this kind of problems. On the other hand, the solution imply solving a traveling salesman problem, so a simple graph query it is not enough, a real algorithm is needed. Our 'light' approach is to address these kind of requirements using scripts, the execution of which, thanks to the flexibility of Eclipse RCP, can be easily integrated into the user interface, e.g. pressing a button. DBin accommodate a module supporting BeanShell [11] scripting language. In Figure 6, one sees a button that has been added by a Brainlet specific configuration of the Map View (a view supporting geo-tagging based on the Google-Earth [12] service). Such button is connected to a BeanShell script that performs the evaluation of the optimal pub tour based on an implementation of a (simple) traveling *salesman* heuristic.



No URI selected Please select a resurce from the views below.

Bannermans Bar
Fiscal
Royal McGregor

Showing semantic clipboard content.

Figure 6: Optimal pub tour around the WWW2006 conference site. A "precooked query" is executed so that certain pubs are selected. The Traveling Pubman script is then invoked to highlight the optimal tour.

4. Conclusions

In this paper we introduced the concept of domain specific Semantic Web applications created using a combination of ontologies, and XML configuration and scripting. Brainlets are a key concept in enabling the creation of Semantic Web Communities in DBin, a full round semantic web platform. We think however that the Brainlet paradigm could be applied to any other semantic web application based on the availability of an underlying SW Database. This could include, in the future, web based Brainlets players capable of providing similarly rich user interfaces over the Web.

While starting a "Semantic Web community" by creating a Brainlet is certainly much more complex than creating a IRC channel or installing a web forum, in our experience domain experts with knowledge of the OWL language have consistently managed to learn all the basic principles, create and deploy their first Brainlets within hours or less. To the best of our knowledge there are no projects directly related to the ability to create Semantic Web access environments which can be compared to the Brainlet paradigm. The idea of a rich client for the Semantic Web has been developed in the Haystack project [13] resulting in a general purpose, monolithic interface. Differently, Brainlets are intended to be powerful user interfaces which can be scripted and 'shaped' on a particular domain. The Piggy Bank project [14] enables light HTML scraping scripts to be installed with ease, so it has, in some sense, a similar philosophy. Also from the MIT Simile initiative, the Fresnel [15] ontology can be used to create "data views" called "lenses" which facilitate the automatic creation of presentation and editing data forms. As such feature are in fact likely to be beneficial to better allow customizations of data presentation, a Fresnel aware Brainlet engine is among the future works.

Brainlets as fostering of social aggregation around Ontologies and data representation practices

When a Brainlet is plugged into DBin, the ontologies that it requires will be installed and used. This simple mechanism, seems to induce a pragmatic, socially based model for ontological agreement, a notably tough problem in Semantic Web research. In fact, as users gather around popular Brainlets for their topic of choice, the respective suggested ontologies and data representation practices will form an increasingly important reality. If someone decided to create a new Brainlet or, in general, a Semantic Web application targeting the same user group as the said popular Brainlet, it is clear that there would be incentive in using identical or somehow compatible data structures and ontologies.

While this is probably the case for the use of ontologies on the Semantic Web in general, such effect is expected to be easy to spot in the real of Brainlets, given how such paradigm immensely lower the barrier by which a domain expert can turn an idea into a full featured domain specific cooperative Semantic Web application.

References

[1] "DBin project" http://www.dbin.org

[2] G. Tummarello, C. Morbidoni, J. Petersson, P. Puliti, F. Piazza , "RDFGrowth, a P2P annotation exchange algorithm for scalable Semantic Web applications" 2004 First P2PKM Workshop, Boston

[3] G. Tummarello, C. Morbidoni, P. Puliti, F. Piazza , "Signing individual fragments of an RDF graph" 2005 WWW2005, poster track

[4] R. Albertoni, A. Bertone, M. De Martino , "Semantic Web and Information Visualization" 2004 Proceedings of the First Italian Workshop on Semantic Web Applications and Perspectives, Ancona (ITALY)

[5] "RDF Gravity - RDF Graph Visualization Tool" Technical Report: HPL-2004-57

[6] E Pietriga , "Isaviz: a visual environment for browsing and authoring rdf models " 2002 11th International World Wide Web Conference

[7] "RDFX" Technical Report: HPL-2004-57

[8] Welkin, a graph-based RDF visualizer, , 2004, http://simile.mit.edu/welkin/

[9] Eclipse Rich Client Platform, , , http://www.eclipse.org/rcp/

[10] Jeen Broekstra, Arjohn Kampman , "SeRQL: An RDF Query and Transformation Language" 2004 ISWC 2004, Hiroshima, Japan

[11] BeanShel: Lightweight Scripting for Java, , , http://www.beanshell.org/

[12] Google Earth, , , http://earth.google.com/

[13] Quan, Dennis and Karger, David R , "How to Make a Semantic Web Browser" 2004 In Proceedings International WWW Conference, New York, USA

[14] D. Huynh, S. Mazzocchi, D. Karger, "Piggy Bank: Experience the Semantic Web Inside Your Web Browser" 2005 proceedings of the ISWC 2005

[15] "Fresnel - Display Vocabulary for RDF" 2005

The Semantics of Collaborative Tagging System

Milorad Tošić and Valentina Milićević

University of Niš,
Faculty of Electronic Engineering,
Intelligent Information Systems Lab – InfosysLab,
Ul. Aleksandra Medvedeva 14, PoBox 73, 18000 Nis,
Serbia and Montenegro
mbtosic@yahoo.com , valentina@elfak.ni.ac.yu

Abstract. In this paper, we adopt a system-oriented approach to the collaborative tagging and define it as a set of interactions in the system of Web resources. First, the system of Web resources is modeled as a set of interacting agents and collaborative tagging is represented as concurrent initiation of interactions between agents in the system. Also, we define concept of knowledge for individual agents. Later we use concepts of interaction and knowledge to give definition of a Link. Then, for a given Universal Set of Resources, we introduce Tag Cloud System (TCS) and definition of (possibly fuzzy) collections of resources. Finally, we introduce concept of Class, based on projection of collections of resources in the TCS, to lay down some of the groundwork towards TCS-based type system.

Introduction

In its essence, Web is all about resource locators (URLs), resource identifiers (URIs) and resource names (URNs) [1] distilling resource as one of the most fundamental concepts of the Web. Until recently, Web was considered only within its original hyper-text framework: web pages are network retrievable text documents, easy to render for human visual consumption, that may contain hyper-links to other web pages. However, massive adoption of the Internet and particularly broadband "last mile", have changed the very nature of the Web that has now been declared "Web as platform". So, Web is not anymore for human eyes only but it is also Web of data. Two different technological and philosophical methodologies are the most visible now days: Semantic Web [2] and Web 2.0 [3]. In spite of the impression that some tension exists between these two communities, we consider Semantic Web and Web 2.0 as two sides of the same coin addressing the same gap between how current technology is applied and the new opportunities. The difference is in the philosophy – general vs. simple: Semantic Web is based on a firm theoretical background and pursues a rigorous, generic top-down approach. In the same time, Web 2.0 is extremely flexible, based on ultimately simple, easy to use and easy to understand stuff, adopts bottom-up approach and worships architecture of participation (services get better as the number of users increases), collective intelligence and long tail model [4].

In [5], authors are concerned with knowledge acquisition for software development, and accordingly they define tagging as chinking and indexing knowledge acquisition dialogue using structures that are relevant to software development. However, the collaborative tagging is more traditionally considered within a framework of strategies that can be used in order to classify and organize content [4,6,7]. The classification strategies are characterized by several distinguishing attributes: If each item may be associated to exactly one category then the strategy is *exclusive*. If each category belongs to a more general one until the root of the tree is attained then the strategy is *hierarchical*. Strategy that is exclusive and hierarchical is called *taxonomy*. One of the typical examples of the taxonomy is the hierarchical directory set up by Yahoo Inc. as an impressive attempt to grow a kind of universal Web taxonomy. Tagging system is a non-hierarchical and non-exclusive strategy where each item is being assigned a list of keywords, called tags. All the tags are at the same level. The tagging systems are further classified by means of who defines the set of words or phrases that may be used as tags and who assigns tags to items. The set of tags may be defined by a central authority, such as editor or a librarian, or may include any word composed of letters. Tags may be assigned to items by the same central authority or by community. For example, in the ACM Computing Classification System [8], the central authority defines the set of keywords that may be used to classify a paper while author of the paper assigns selected keywords to the paper[1]. *Collaborative Tagging (Folksonomy)* is an ad hoc classification scheme that Web users invent as they surf to categorize the data they find online. Consequently, it is *anarchic* (the choice for the keywords are not restrained by any central authority but may be any string of alphanumeric characters) and *democratic* (the tagging is performed by a large ensamble of people, and not by a central one) [7]. Social software – software that enables users to share information and collaborate online – makes these tags available to other users, who can than take advantage of all this tagging to search for the information they need [4]. This approach has become increasingly popular, and some Web sites (call them Web 2.0 or not?) maintain ***tag cloud***, a list of all tags used on the domain usually with a visual indication of individual tag's popularity. The *collaborative filtering* is a democratic method of classification that does not require tags to be words only. The collaborative filtering exploits user access patterns to link items to people who use it [7].

We can identify three orthogonal dimensions of the concept of *scripting language*: 1) *Language* characteristics that identify a programming language as a scripting language (weak typing or even no typing at all, reflection and introspection, etc.); 2) *System* that is programmed by the scripting language (in the case of OS shell scripting languages the system is set of OS commands, while in the case of MSVisualBasic the system is composed of a set of registered ActiveX and/or COM components); and 3) *Application* under development. In this paper, we are focused on theoretical foundations for the second aspect, i.e. we envision Tag Cloud System as a system that will

[1] To the best of our knowledge, there is no tagging system like ACM CCS where set of possible keywords is defined by a central authority while readers assign the list of tags (or at least are allowed to edit it) to the paper instead of author of the paper.

be programmed by future semantic scripting languages in order to develop whole new set of global scalable applications for the "Web as a platform". Introduction of semantics into the traditional scripting brings in two additional levels of freedom: 1) Using existing scripting languages to develop semantic applications (e.g. JavaScript programs a client side while Ruby on Rails, PHP and scripting language for programming plug-ins in Wiki are on a server side), and 2) Using "semantic scripting language" to develop not exclusively semantic applications, but also traditional applications, such as CMS for example.

In this paper, we consider collaborative tagging in a way that addresses the problem on today's Web of bridging the gap between wide adoptability, easy to use, and simplicity from one side, and ability to address problems in a general way by adoption of the formal foundation. First, the system of Web resources is modeled as a set of interacting agents. We adopt a system-oriented approach to the collaborative tagging and define it as a set of interactions in the system of Web resources. Also, we define a concept of knowledge for individual agents based on their local state. Later we use concepts of interaction and knowledge to give definition of a Link. In the third section, for a given Universal Set of Resources, we introduce Tag Cloud System (TCS) and definition of (possibly fuzzy) collections of resources. In the fourth section, we introduce concept of Class, based on projection of collections of resources in the TCS. In this way, we lay down some of the groundwork towards TCS-based type system. Finally, we discuss a few pointers for future work and give some concluding remarks.

Semantics of the Concept of Resource

Debate over the concept of resource

In the early days of the Web, semantics of the resource concept has been much less important comparing to application and adoption aspects of the concept. As a natural consequence, the concept of resource was traditionally comprehended as *a network 'retrievable' entity*. However, mass adoption of the Web has resulted in completely new understanding of the value of the Web. For example, Semantic Web is one of the most promising candidate prospects. For the Semantic Web, understanding of the concept of resource is of the paramount importance because transferring data is not enough any more: Now, we have the need to communicate knowledge. To do so, we have to move up the ladders of abstraction, adopt a higher meta level as an operational level, and manipulate with knowledge and interaction instead of data and communication. Having that in mind, it is somewhat surprising that there is still an ongoing debate over definition of the resource in the literature as well as in the community [1,7,10,11,12].

Although there is a stated definition of a resource in the URI RFC, it is in many respects vague: "*A resource can be anything that has identity*. Familiar examples include an electronic document, an image, a service (e.g., 'today's weather report for Los Angeles'), and a collection of other resources. *Not all resources are network 'retrievable'*; e.g., human beings, corporations, and bound books in a library can also be considered resources. *The resource is the conceptual mapping to an entity or set of entities*, not necessarily the entity which corresponds to that mapping at any particular instance in time. Thus, a resource can remain constant even when its content – the entities to which it currently corresponds – changes over time, provided that the conceptual mapping is not changed in the process." [13].

The ongoing debate about the difficult problem of semantics of the concept of resource is very important, probably should receive a stronger support from at least one official standardization organization, and involves very diversified and heterogeneous scientific disciplines. In this paper, we do not want to get involved into the debate being aware of possible inconsistency in the rest of the paper. Instead, we give the following statement, based on [13,14], and consider it as correct enough for the purpose of the paper:

Resource is a generic term for anything in the universe of discourse that has identity.

Though, having in mind very limited implementation value of the statement [1,7,10,11,12], we allow further refinements in the rest of the paper on as needed bases. Comparing our previous statement about resource to the definition given by WordNet [15] that a resource is "a source of aid or support that may be drawn upon when needed (*the local library is a valuable resource)*" we may say that, by our statement, knowledge about identity of anything in the universe of discourse has a value on its own.

Multi-agent interpretation

In our system, there are two first-class meta-classes of objects: 1) *Resource* and 2) *Link*. All further constructs are built upon these two meta-classes of objects. As a modeling foundation for the definitions of the resource and link concepts, we adopt an approach that follows distributed knowledge theory developed by Joseph Halpern[2], particularly work on knowledge-based protocols [16]. In the following, we introduce basic system modeling concepts using body of work from [16] as a foundation. However, we use the concepts introduced that way in substantially new manner such that they provide foundation for presenting some of our original ideas, particularly ones related to the concept of interaction.

Let us given set of entities $AG=\{ag_i|\ i=1,2,\ldots,n\}$, called *agents*, such that each agent in the set carries certain amount of its own local information. The agent may change its local information and any change of the local information is observable by the agent. The local information is also called the agent's *local state, $s(ag_i)$*.

[2] See http://www.cs.cornell.edu/Info/People/halpern/abstract.html for the complete list of his work.

Definition 1: The set of agents, $AG=\{ag_i| \ i=1,2,\ldots,n\}$, that may ever exist in any system under consideration is called an *universal space of resources U*, also referred to as the *universe of discourse*.

We consider the given set of agents $AG=\{ag_i| \ i=1,2,\ldots,n\}$, to be closed: There is one agent in the set, called *environment*, that models state and interactions that are out of scope of the modeled system. In other words, there is no agent outside the set of agents that any agent from the set interacts with, ever: $(\forall ag_i \in AG) \ ag_i \bullet ag_j \Rightarrow ag_j \in AG$ .Note that the set of agents is not considered closed on its own sake. Instead, it is closed with respect to the modeled system. In other words, the set of agents represents our knowledge about the modeled system. Also, it represents structure that we use to reason about the system.

If agent ag_i may change local state of some other agent ag_j, or if agent ag_j may observe a (certain) change in the state of the agent ag_i, then we say that agents ag_i and ag_j are *interacting*, and such their setting is called *interaction,* ρ_{ij}*:* $ag_i \bullet ag_j$. Let us denote the set of all interactions between agents from the set AG as $R_{AG}=\{\rho_{ij}| \ ag_i \ , ag_j. \in AG\}$. We also say that an *interaction protocol* is initiated between two interacting agents. Any single agent may be involved into zero, one or more interactions. Part of the local state $s(ag_i)$ that may be changed by the agent ag_j or that is observed by the agent ag_j within the interaction ρ_{ij}, is called *projection of the local state on the interaction*, and denoted as $s(ag_i)\downarrow \rho_{ij}$. Union of projections of local state $s(ag_i)$ on all existing interactions, $s^O(ag_i) \ = \ \cup \, s(ag_i)\downarrow \rho$ *for all* $\rho\in R_{AG}$, is called *observable part of the local state*. *Local state of the interaction* ρ_{ij} , denoted as $s(\rho_{ij})$, is defined as a union of projection of the local state on the interaction for each agent involved into the interaction, $s(\rho_{ij})= s(ag_i)\downarrow \rho_{ij} \ \cup s(ag_j)\downarrow \rho_{ij}$. An agent ag_i is called *passive (active) with respect to interaction* ρ_{ij}*,* if it cannot (may) change local state of the interaction. A *passive agent* is an agent that is passive with respect to all existing interactions. An agent that is not passive is called *active agent*. Observable part of local state of a passive agent can be changed only as a consequence of interaction with an active agent. However, we say nothing about non-observable part of the local state – meaning that an agent may change non-observable part of its state and still be considered as a passive agent.

For example, intended semantics of the interaction may be illustrated by means of a shared variable between two concurrent threads, where threads represent agents, *ThrdA* and *ThrdB*, and shared variable represents the interaction. If the shared variable is part of local state of each of the interacting agents then each of the threads is modeled as an active agent. However, we can consider the shared variable to belong to the local state of only one agent. In that case, the agent having the shared variable as part of its local state may be modeled as passive or active, while the other agent must be active (if the other agent is not active then there would not be any interaction). Note also that the interaction is about change in a state but not about data transfer as is the case with a communication protocol. The important difference between interaction and communication protocols is in the level of abstraction where the change happens: In the case of data transfer, the change is always in the value of data. However, in the case of interaction, the change can be in data, information, knowledge, or some other interpretation. Note that different protocols represent different kinds of possible interactions between agents. Also, different interaction protocols may be interpreted at different meta levels. Now, let us examine the case where the two threads communicate some data from *ThrdA* to *ThrdB* in a send-receive fashion such that *ThrdA* is sending while *ThrdB* is receiving data. The communication protocol may be like this: Initially, value of the shared variable is zero; *ThrdA* sets new value in the shared variable; *ThrdB* probes value in the shared variable permanently, and when the value is not zero *ThrdB* copies the value into some other place in its local state; After reading the value, *ThrdB* set value of the shared variable to zero; After setting new value, *ThrdA* has started to probe the value; After registering zero value in the shared variable, *ThrdA* knows that it is safe to set next value. In this case, at the interaction level both agents are active because each of them changes value of the shared variable. However, at the communication level, we say that the sender (*ThrdA*) is active while the receiver (*ThrdB*) is passive.

Definition 2: *Body of Knowledge (BK)* of an agent ag_i is defined as a part of its local state that is not observable $BK_i \equiv s(ag_i)/s^O(ag_i)$.

Definition 3: *Link* is knowledge that an agent has about identity of some other agent. The link is knowledge that is sufficient for the agent to initiate an interaction protocol with the linked agent.

Definition 4: Let us given an interaction protocol, set of agents (called resources) and an individual agent (called agent) such that the agent can interact with the resources by the given protocol. *Addressing* (or *Code*) of the set of resources is a common service, such that there is guaranty that if an agent encounters the interaction protocol with different end addresses then it will interact with different agents, i.e. it may eventually experience different interaction histories.

In order to give an example for the previous definitions, let us consider an agent ag_{new} that has just been introduced into the universe of discourse. Since ag_{new} doesn't have any interaction history, it has empty body of knowledge, $\mathrm{BK}_{\mathrm{new}}=\varnothing$. Because $\mathrm{BK}_{\mathrm{new}}=\varnothing$, agent ag_{new} doesn't know about any links and is not able to activate any interaction. It has to wait for some other active agent to initiate interaction with the new agent. After finishing an interaction, it is expected that ag_{new} may have remembered[3] something from the previous interaction such that it may now have $\mathrm{BK}_{\mathrm{new}}\neq\varnothing$. If ag_{new} have learned address of some other agent during its last interaction, then ag_{new} may be able to initiate interaction with the agent on that address.

[3] An agent may or may not remember interaction histories depending on its internal memory resources. However, taking this into consideration is definitely out of scope of the paper.

Tag Cloud System

In this section, based on the previously defined concepts of resource, link, interaction, and knowledge, we present our understanding of the collaborative tagging as a set of concurrent acts of introducing new links into the system.

Definition 5: *Tag Cloud (TC)* is a tuple $TC = (R,L)$ where $R \subset U$ is a non empty *set of resources* contained in an *universal space of resources* U, also referred to as the universe of discourse, $L = \{(r,RID(p)) \mid r \in R, p \in U\}$ is a set of links, $RID\ (p): R \rightarrow A$ is a *resource identity function* that is mapping from the set of resources to the set of addresses A.

Note that the previous definition introduces a purely abstract category of resource as a member of the set of resources R and by means of the resource identity function *RID*. We say nothing about what the resource is, what is it's nature, structure, behavior or else. At this point, there is no semantics assigned to the resource. Instead, the resource can be anything that participates as a source of a link in the *TC*. The set of addresses A may be subset of a language or a subset of an enumerated set. The fact that set of addresses A is a subset of a language (or enumerated set) should be interpreted such that not every correct language construction is an address in the *TC*.

The *TC* represents a distributed knowledge system in a sense that we may consider something as a resource only after we learned about it as a resource. Similarly, we may consider a correct language construction as an address only after we learn about it as an address of a resource in the *TC*. On the other hand, the only way we can learn about new resources is to interact (inspect) with resources that are participating in our current knowledge. Further, In other words, we cannot speculate about anything that is not linked to at least one resource from the *TC*. In that way, we may say that *TC* represents the *Resource Universe*.

The natural interpretation of the TC is set of agents, as is introduced in Definition 1. We indicate this fact by the requirement that set of resources in the tag cloud is subset of the universe of discourse. In this way, we apply developed semantics of the multi-agent system to the tag cloud.

Definition 6: *Tag Cloud System (TCS)* is a tuple $TCS=(R,L,\Sigma)$, where $TC = (R,L)$ is a Tag Cloud, and Σ is a set of collections of resources from U such that each *collection* $C \in \Sigma$ is defined by the associated membership function *mC*.

The Tag Cloud System is a fine extension of the Tag Cloud structure that allows us to introduce collections into the Tag Cloud. The collection is defined by means of its membership function, with no constraints made on the function. The idea here is to have flexibility to being able to introduce different collections with membership functions of different nature, including fuzzy sets [17,18]. For example, in order to define set of tags R as a collection in the universe of discourse U, we use the membership function from the classical set theory: $mR: U \rightarrow \{0,1\}$, where $\forall u \in U, mR(u)=1$ if $u \in R$, and $mR(u)=0$ otherwise. However, we are not constrained to use such classical (or crisp) sets only. We can also use fuzzy set, which is a more general concept then the classical set: The membership of an element to a fuzzy set is not described by a Boolean function (as it is the case for a classical set), but by real values between 0 and 1, in general [18] (note that it can also be any other function, including discrete functions).

As we mentioned before, in the interpretation of the TCS, one cannot reason over anything else other then agents knowledge BK_i. Consequently, the membership function for an agent ag_i must be defined over BK_i only, for any collection under consideration. Typically, the BK_i includes addresses of other agents that are pointed to by links starting at agent ag_i.. However, we do not put any restriction on the type of knowledge that may constitute BK_i. Thus, in addition to the "network topology" knowledge, an agent may have a free form text (for example, comment of the user who has created the agent while tagging some information on the Web), pictures, and any other type of structured or un-structured data. Later on, this knowledge is used for search or information extraction or any other purpose. The important advantage is that we integrate network topology information and free form information such that it can be queried in a unified manner.

Now, we introduce projection between two collections as a binary operation $\downarrow$ in the set of collections of a TCS.

Definition 7: Collection $C=C1\downarrow C2$ is *projection* of the collection $C1$ on the collection $C2$, defined as $C=\left(C1\downarrow C2\right)\equiv\left(\bigcup_{o\in C1}Chdrn(o)\right)\cap\left(\bigcup_{o\in C2}Chdrn(o)\right)$, where $\cup$ is (fuzzy) union and $\cap$ is (fuzzy) intersection operations

The projection of two collections, is defined with an aim to capture semantics of the TCS in the following way: First, find all tags of all resources from *C1*. Then find all tags of all resources from *C2*. Finally, find the intersection of the two sets of tags. The resulting set of tags should interpret "similarity" between collections *C1* and *C2*.

Resource Class in the TCS

Traditionally, we define class as *a collection of objects featuring some common (set of) feature(s)*. Following the previously introduced definitions, we may introduce class into the TCS in a similar way:

Definition 8: ***Class*** *C* in *TCS=(R,L)* is tuple *(O,T)* where $O\subseteq R$ is a collection of resources, called ***objects***, and $T\subseteq R$ is a collection of (meta)resources, called ***tags*** (or features), such that every object $o\in O$ has identical projection of the collection of it's children into the given collection *T:* $\forall o1,o2\in O\ (Chdrn(o1)\downarrow T)=(\ Chdrn(o2)\downarrow T).$

More informally, the set O is interpreted as a set of objects belonging to the class C. Set T is a subset of the set of all resources (resource universe) such that it's elements are identifiable as assigned semantics of being features. In other words, set T is subset of tags. However, we have to have a method to identify single resource as a tag. We implement this identification such that we have defined the page *Tag* with assigned semantics that *every resource R that has incoming link from the page Tag is perceived as being a tag*. The page *Tag* has a link to itself meaning that it is also tag.

Conclusion, Application Aspects and Future Work

To the best of our knowledge, the work presented in this paper introduces a theoretical model for semantics of the collaborative tagging systems, for the first time. We set the foundation for further exciting developments, particularly towards overcoming the gap between tagging as a Web 2.0 and tagging as a Semantic Web. The underlying model of knowledge-based multi-agent system has proven to be very helpful for us in solving practical application problems that show up during development of our tagging application prototype. In the prototype[4], we adopt and implement *Resource* and *Link* concepts. In that way, we got the unified, technology transparent, Semantic-Wiki-Tagging system. For example, according to Definition 3, each tagging contains link to the date when this tagging has been performed. However, we do not need to create an actual Wiki page for every such a date: agents in the system ('Wiki pages containing tagging data') have knowledge about identity of the date in a form of a Link. From the other side, the only interaction that may be initiated with the date is 'create page' because the date page is not able to engage into 'view page' interaction. Our future research will be to address the theoretical formulation of similar issues of the working prototype in more details.

The short indication given in the last section is particularly promising for future research. Definition of a tagging framework, similar to Object Oriented Programming, would definitely empower a whole new application space. One of the future challenges would be an object behavior within the TCS semantics. It is an open question whether a collection of resources is a resource itself (has an address or URI) or not. The similar problem exists with blank nodes in RDF [14,21]. Hence, we expect solutions similar to the one presented in [21] to be effective in the case of TCS too.

In this paper, we introduced the theoretical foundation for addressing different aspects of the semantic scripting by considering the collaborative tagging as a low-level scripting language on the global computational services fabric called "Web as a platform"[5]. We were focused on the system aspect of the semantic scripting. Depending on the level of abstraction[6], the target application may be traditional (collaborative bookmarks, annotations, etc.) or semantic (semantic Wiki, semantic web portal, semantic e-mail, etc.) or something completely innovative and new (such as tag clustering, tag hierarchies, tag cloud management, weighting and sequencing of tags, etc.).

[4] Code base of the prototype initially started as a modification of JSPWiki open source Wiki engine [19]. However, in time they developed into two almost independent applications.

[5] Analogous to shell scripting on an OS platform

[6] Level at the semantic web stack [20]

We are developing several Web applications based on collaborative tagging paradigm described in this paper. The current tagging application prototype can be accessed for testing at http://infosys-work.elfak.ni.ac.yu/InfosysWiki-v2-1/Wiki.jsp?page=TagCloud

References

1 Dan Connolly, Untangle URIs, URLs, and URNs: “Naming and the problem of persistence,” *IBM developerWorks,* http://www-128.ibm.com/developerworks/xml/library/x-urlni.html

2 Tim Berners-Lee, James Hendler and Ora Lassila, “Semantic Web”, *Scientific American*, May 2001.

3 Tim O'Reilly, “What Is Web 2.0, Design Patterns and Business Models for the Next Generation of Software,” Sept. 30. 2005, Oreillynet.com, http://www.oreillynet.com/pub/a/oreilly/tim/news/2005/09/30/what-is-web-20.html

4 McFedries, P., “Folk Wisdom”, IEEE Spectrum, Feb. 2006., pp.80-80.

5 Heather Richter, Chris Miller, Gregory D. Abowd, and Harry Funk. “Tagging Knowledge Acquisition To Facilitate Knowledge Traceability,” in *Proceedings of the Conference on Software Engineering and Knowledge Engineering (SEKE)*, July 2003, pp432-439.

6 Golber, S., Huberman, B.A., “The Structure of Collaborative Tagging System”, Information Dynamics Lab: HP Labs, Palo Alto, USA, available at: http://arxiv.org/abs/cs.DL/0508082

7 Lambiotte, R., Ausloos, M., “Collaborative tagging as a tripartite network”, arXiv:cs.DS/0512090 v2 29 Dec 2005.

8 ACM Computing Classification System, http://www.acm.org/class/1998/

9 Champin, Pierre-Antoine; Jérôme Euzenat; Alain Mille: "*Why URLs are good URIs, and why they are not*", May 2001, http://www710.univ-lyon1.fr/~champin/urls/

10 Adam Mathes, “Source vs. Resource Ontology” http://www.adammathes.com/academic/krfo/sourceresource.html

11 Clark, Kendall Grant: “Identity Crisis,” *XML.com*, September 11, 2002, http://www.xml.com/pub/a/2002/09/11/deviant.html

12 S. Pepper and S. Schwab. “Curing the Web's Identity Crisis,” *Technical report, Ontopia* http://www.ontopia.net, 2003.

13 Berners-Lee, T.; R. Fielding; L. Masinter: "Uniform Resource Identifiers (URI): Generic Syntax and Semantics", *RFC 2396*, August 1998, http://www.ietf.org/rfc/rfc2396.txt

14 Hayes, P., “RDF Semantics – W3C Recommendation”, http://www.w3.org/TR/rdf-mt/, 2004.

15 Wordnet, http://www.cogsci.princeton.edu/~wn/

16 Ronald Fagin, Joseph Y. Halpern, Yoram Moses, and Moshe Y. Vardi, “Knowledge-based programs,” *Distributed Computing,* 10:4, 1997, pp. 199-225., also at http://www.cs.cornell.edu/Info/People/halpern/abstract.html

17 L.A. Zadeh. Fuzzy Sets. *Information and Control*, 8:338--353, 1965.

18 Florea, M.C., Jousselme, A-L., Grenier, D., Bossé, É., “Combining belief functions and fuzzy membership functions,” Multisensor, Multisource Information Fusion: Architectures, Algorithms, and Applications 2003. Edited by Dasarathy, Belur V. *Proceedings of the SPIE*, Volume 5099, pp. 113-122, 2003.

19 http://www.jspwiki.org

20 Tim Berners-Lee. Web for real people, 2005. Available at http://www.w3.org/2005/Talks/0511-keynote-tbl/.

21 Carroll, Jeremy J.; Bizer, Christian; Hayes, Pat; Stickler, Patrick, “Named Graphs”, *Journal of Web Semantics,* Vol. 3, No. 4, 2005, available at http://www.websemanticsjournal.org/ps/pub/2005-23

Semantic Scripting Challenge Submissions

RDFHomepage[⋆]

Gunnar AAstrand Grimnes, Sven Schwarz, and Leo Sauermann

Knowledge Management, DFKI GmbH
Kaiserslautern, Germany
http://www.dfki.uni-kl.de/∼{grimnes,schwarz,sauermann}
{grimnes,schwarz,sauermann}@dfki.uni-kl.de

Abstract. This paper presents the RDFHomepage project, a framework for using a person's structured data sources to auto-generate an HTML homepage. RDFHomepage uses RDF files as input, and currently supports several well-known RDF schemas, such as FOAF. In addition to these we have RDF converters for other structured file-formats, like Bibtex. RDFHomepage produces valid HTML 4.01 Transitional pages, and makes it easy to roll-out functional homepages for a group of people. The generated HTML code is very general, allowing quick and easy page-redesigning using CSS. RDFHomepage is written in PHP and uses our system for generating PHP classes based on RDF class definitions, enabling quick and easy development of RDF handling PHP code.

1 Introduction

RDFHomepage is a tool for automatic generation of HTML homepages based on RDF files and other structured information sources which a user might already create and maintain. The generated page contains all things one expects on a typical homepage: it shows the person's name, email, telephone number etc., this is taken from his FOAF profile; there is a short bibliography, this taken from a file using the *homepage-schema* we created for this task; there are sections listing the projects a user is involved in, taken from the DFKI Organisational Repository (OrgRep); and section with the user's research interest. RDFHomepage generates pages that are valid HTML 4.01 Transitional, with HTML code is well structured and can easily be styled with CSS. A part of RDFHomepage is an engine for generating PHP template classes from RDFS Class descriptions. This has two main advantages, it removes the need to know the details of RDF processing, and it enables code-completion, removing the need to know the details of the schema being used.

2 Architecture

For RDFHomepage we chose to use the web-scripting language PHP. We chose PHP because it is free and open-source, and is a powerful and feature complete

[⋆] This is a shorter version of a paper submitted to the Semantic Web Scripting Workshop language, with good support for RDF through the RDF API for PHP (RAP) [1]. RDFHomepage uses RDF data from several standard sources, detailed below, and we also created a homepage schema, providing the semantic glue between these sources, and allowing the user to specify additional personal details in a structured form, for example his interests or personal views on projects.

2.1 RDF Data

FOAF The Friend-of-a-Friend ontology [2] was the main point of inspiration for RDFHomepage. A huge number of people in the semantic web community have created their own FOAF profile and published it[3], and there are millions more generated by LiveJournal[4], Ecademy[5] and other social sites producing FOAF.

Bibtex BibTeX is a format for managing citations when using TeX or LaTeX. BibTex defines different classes of publications, such as articles, books, theses, etc., and associated optional and required properties of these. Most computer scientists will keep a BibTeX file of their own publications up to date, for use when self-citing or when publishing their papers on their website. Since people already maintain this information in a structured format it makes sense to reuse it, and to this end we used BibTeX2RDF, written by Wolf Siberski[6].

Projects At the DFKI information about all projects and the people working in them is centrally maintained in a Organisational Repository (OrgRep)[7]. OrgRep was originally created for use in the FRODO project, but has been maintained and used in many DFKI projects since, for example EPOS and MyMory.

3 Conclusion

RDFHomepage provides a quick and easy to way to create attractive and informative homepages, with all the content typically found on a research active user's homepage. The pages are generated from RDF files allowing complete separation of content and appearance, as well as making the homepage machine processable by semantic web agents. RDFHomepage is open source, release under a GNU Public License (GPL) and can be downloaded from http://rdfhomepage.opendfki.de. To get a better impression of what RDFHomepage can do, consider visiting any of the author's homepages, they are all automatically-generated![8]

[1] http://www.wiwiss.fu-berlin.de/suhl/bizer/rdfapi/
[2] http://xmlns.com/foaf/0.1/
[3] http://rdfweb.org/topic/FOAFBulletinBoard
[4] http://www.livejournal.com
[5] http://www.ecademy.com/
[6] http://www.l3s.de/∼siberski/bibtex2rdf/
[7] The OrgRep ontology: http://www.dfki.uni-kl.de/∼grimnes/2006/03/orgrep/orgrep.rdfs
[8] See also http://rdfhomepage.opendfki.de/cgi-bin/trac.cgi/wiki/WorkingInstallations

RDFRoom – In an Angular Place*

Gunnar AAstrand Grimnes

Knowledge Management, DFKI GmbH
Kaiserslautern, Germany
http://www.dfki.uni-kl.de/∼grimnes/2006/03/RDFRoom
grimnes@dfki.uni-kl.de

Abstract. A lone soldier has been stranded in an alien world, filled with resources, literals and shifty anonymous nodes. Room upon room are filled with named graphs – can he find a way out?
RDFRoom is an isometric RDF viewer. It gives the user ways to view and manipulate his RDF data that might make him see the data in a brand new perspective.

1 Motivation

The ideas for RDFRoom came together from many different sources of inspiration. Most recently was Danny Ayers' post about the *Web of World Craft*[1], where he speculated that Web is best suited for displayed document or database "shaped" data. Since RDF can go beyond that, and describes things, not just data, the current World-Wide-Web paradigm will always have trouble displaying such graph-based information, and games might be a better candidate for representing such information.

An earlier source of inspiration was a conversation with colleagues in the DFKI Knowledge Management lab, where it was noted that finding things in a computer-game is much easier than in the folder-structure on your harddisk. Personally I reorganise my folders ever so often, but I still lose documents quite regularly. If I could put my files in a 3D world I know well – I would always remember that the PDF of my CV goes under the stair by the rocket launcher, and last year's tax-returns go with the mega-health.

Two pieces of previous work were also crucial for the ideas of RDFRoom. Firstly, Dennis Chao's psdoom [1], a version of Doom where processes running on a machine appeared as monsters in a doom level and killing them meant killing the process. His paper also contains several good observations about the effects of using such a user-interface. Secondly, Liam Quinn wrote an RDF based adventure game (RDFG)[2] where the world and the objects in it are described in RDF. In Liam's game the world was carefully scripting beforehand, but in RDFRoom the world is the web itself, although the game is admittedly more pointless than RDFG.

[*] Thanks to Danny Ayers for the title

[1] http://dannyayers.com/2006/03/26/web-of-worldcraft

[2] http://dirk.holoweb.net/∼liam/rdfg/rdfg.cgi

A third source of inspiration was the IRC Client Colloquy[3], which plays a shotgun sound effect when people are kicked from a room. This makes for very satisfying and immediate feedback and does also highlight the seriousness of the action. We wanted RDFRoom to mimic this feedback when deleting nodes. Put together with the recent interest in RDF browsers and browsable data [2], these things made me spend a few evenings coding RDFRoom.

2 Overview

RDFRoom represents a graph as room in an isometric world. One node of the graph is used as the starting point - this is either specified by the user, or RDF-Room will pick the node with the most out-going edges if no node is specified. Each node is represented by an object in the world, there are default "blobs" for resources, literals and anonymous node, and to make the world more interesting typed objects have appropriate graphics, for example, *mailto:* URIs are shown as e-mails, *foaf:Person* as characters, etc. Additional types can be added through RDF style-sheets. The player can walk around in the world, and inspect and re-arrange the nodes. *rdfs:seeAlso* links are used to generate door to other rooms, representing other graphs. Door where the *seeAlso* link cannot be loaded, because of HTTP errors, etc. will be shown as closed.

3 Technical Details

RDFRoom was implemented in about 10-15 hours, using Python, PyGame[4], the fantastic RDFLib[5], and the isometric engine Isotope[6], modified to allow an RDF based world.

4 But Why?

That is a very good question, and RDFRoom might be a good candidate for the most useless code I've ever written. However, the question could equally well be "Why not?", and at least I had fun in writing RDFRoom. I also think there is at least traces of a serious message here, and maybe it will make people think of different ways to view data than the document paradigm.

References

1. Chao, D.: Doom as an Interface for Process Management. In: Special Interest Group on Computer-Human Interaction. (2001)
2. Berners-Lee, T.: Browsable data. Invited Talk (2006) http://www.w3.org/2006/Talks/0302-browsedata-tbl/.

[3] http://colloquy.info

[4] http://www.pygame.org

[5] http://rdflib.net

[6] http://www.webalice.it/simon.gillespie/Isotope.html

A prototype for faceted browsing of RDF data

Eyal Oren and Renaud Delbru
DERI Galway
firstname.lastname@deri.org

May 15, 2006

1 Faceted browsing

Faceted browsing is a superior exploration technique for large structured datasets [4], useful when users do not know the data that they are looking for. In faceted browsing, the information space is partitioned using orthogonal conceptual dimensions of the data (called facets). Each facet has multiple restriction values; users select a restriction value to constrain relevant items in the information space.

Existing approaches such as Flamenco [4] and Ontogator [1], cannot navigate arbitrary datasets but are limited to manually defined facets over predefined data structures. We have defined an approach for automatic construction of facets in semi-structured RDF data [3]. We extend the notion of faceted browsing to (graph-based) RDF data and define metrics for automatic ranking of facet quality. Our technique works for arbitrary RDF data, without the need to conform to any schema.

In this paper we present the accompanying prototype implementation.

2 Prototype

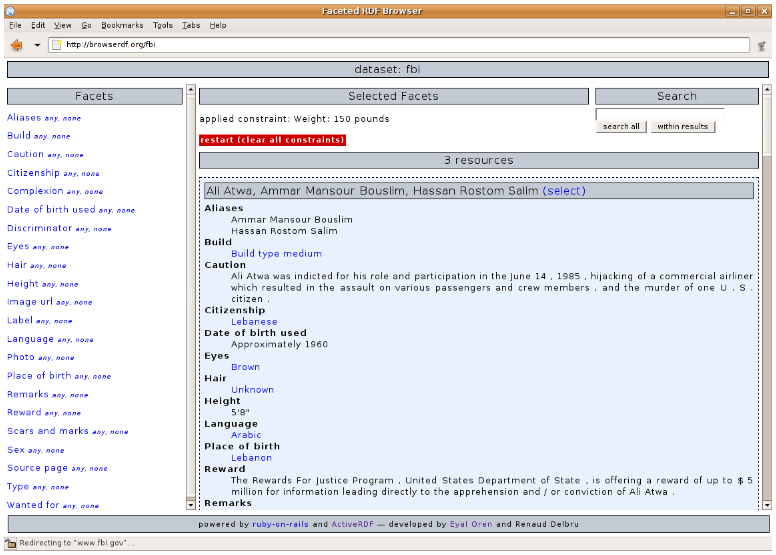
The prototype implementation is available at `http://browserdf.org`. A screenshot, showing an RDF dataset[1] representing the FBI most-wanted fugitives, is shown in Fig. 1. The interface is automatically generated for this data, and works similarly for arbitrary data.

On left-hand side of Fig. 1 we see the various facets in the dataset, e.g. alias, eye-color, hair-color, or nationality. The user can select any facet and restrict it, either by directly choosing restriction values (for literal values) or by indirectly (for resources which themselves have properties that are facets). At the top the currently applied constraints are shown, and the main part displays the resources that conform to the current constraints.

The user is prevented from following dead-ends: at each step, only those facets and restriction values that have non-empty results are available. Searching with keywords is also available, either within the current selected resources or in the whole set.

The screenshot shows a simple restriction-value over the weight of a person. The prototype offers several other types of constraints, such as arbitrary joins, arbitrary selection focus, property inversion, and intersection of these.

[1] `http://sp02.stanford.edu/kbs/fbi.zip`



dataset: fbi

Facets

Aliases *any, none*
Build *any, none*
Caution *any, none*
Citizenship *any, none*
Complexion *any, none*
Date of birth used *any, none*
Discriminator *any, none*
Eyes *any, none*
Hair *any, none*
Height *any, none*
Image url *any, none*
Label *any, none*
Language *any, none*
Photo *any, none*
Place of birth *any, none*
Remarks *any, none*
Reward *any, none*
Scars and marks *any, none*
Sex *any, none*
Source page *any, none*
Type *any, none*
Wanted for *any, none*

Selected Facets

applied constraint: Weight: 150 pounds

restart (clear all constraints)

Search

3 resources

Ali Atwa, Ammar Mansour Bouslim, Hassan Rostom Salim (select)

Aliases
Ammar Mansour Bouslim
Hassan Rostom Salim
Build
Build type medium
Caution
Ali Atwa was indicted for his role and participation in the June 14 , 1985 , hijacking of a commercial airliner which resulted in the assault on various passengers and crew members , and the murder of one U . S . citizen .
Citizenship
Lebanese
Date of birth used
Approximately 1960
Eyes
Brown
Hair
Unknown
Height
5'8"
Language
Arabic
Place of birth
Lebanon
Reward
The Rewards For Justice Program , United States Department of State , is offering a reward of up to $ 5 million for information leading directly to the apprehension and / or conviction of Ali Atwa .
Remarks

powered by ruby-on-rails and ActiveRDF — developed by Eyal Oren and Renaud Delbru

Redirecting to "www.fbi.gov"...

Figure 1: Faceted browser showing FBI data

3 Implementation

The prototype is implemented using the web application framework Ruby-on-Rails[2] and the object-oriented RDF API ActiveRDF[3] [2]. It works on arbitrary RDF data sources through ActiveRDF datastore adapters: the YARS adapter is most suitable since it supports aggregation queries and keyword queries; arbitrary SPARQL datasets can be displayed as well, but are relatively slow due to the lack of aggregation queries.

We are currently performing an comprehensive evaluation, about which we will report in future work. Preliminary evaluation results look promising: the metrics give priority to the most important predicates, the automatically constructed interface gives a usable overview of the dataset and users find the interface highly usable.

Acknowledgements This material is based upon works supported by the Science Foundation Ireland under Grants No. SFI/02/CE1/I131 and SFI/04/BR/CS0694.

References

[1] E. Hyvönen, S. Saarela, and K. Viljanen. Ontogator: Combining view- and ontology-based search with semantic browsing. In *Proceedings of XML Finland*. 2003.

[2] E. Oren and R. Delbru. ActiveRDF: Object-oriented RDF in Ruby. In *Scripting for Semantic Web (ESWC)*. Jun. 2006.

[3] E. Oren, *et al.* Annotation and navigation in semantic wikis. In *SemWiki in ESWC*. 2006.

[4] K.-P. Yee, K. Swearingen, K. Li, and M. Hearst. Faceted metadata for image search and browsing. In *CHI*. 2003.

[2] http://rubyonrails.org
[3] http://activerdf.org

FOAFMap: Web2.0 meets the Semantic Web

Alexandre Passant

Laboratoire LaLICC
Université Paris IV, France
alex@passant.org

Abstract. FOAFMap - `http://foafmap.net` - is an online service providing geolocation with a FOAF and Google Maps mashup, as a mix of both Semantic Web and Web 2.0 technologies.

1 Motivations

Web 2.0 geolocation services such as Frappr![1] allow users to create, or subscribe to, groups and related maps.

FOAFMap's idea is to provide an equivalent service using decentralized profile and data management, thanks to Semantic Web principles and FOAF[1] vocabulary. Thus, people manage themselves their data while the tool just reads, understands and displays it in an appropriate way.

2 Overview and Implementation

FOAFMap users don't need to register, but just have to provide a FOAF file URL, which could point to either a personal profile or a group document.

After identifying the document type (personal profile or group), the service retrieves people referenced in the file. For each people found, the script parses his personal FOAF file - if any - and extracts geolocation information that he may have provided with Geo Vocabulary[2]. Then, it displays these people on a Google Map, with other personal information: name, weblog or email, and even a resized picture if available on the profile.

As parsing the profile and retrieving files can be really long when there is a lot of people referenced in it, FOAFMap allows users to cache the created map in its filesystem, so that it can then be displayed faster.

Since tagging became a common practice in Web2.0 services, FOAMap also allows people to tag their maps, but once again, tags are not created locally but extracted from FOAF files. Actually, the retrieved tags are `foaf:interest` properties of the user or group mentioned in the file. Tags are identified with their URL, and FOAFMap provides a way to see aliases of the same tag, as anyone can provide the title he wants using `dc:title`. Like most of the tools using folksonomies, FOAFMap allows people to see who shares common tags,

[1] `http://frappr.com`

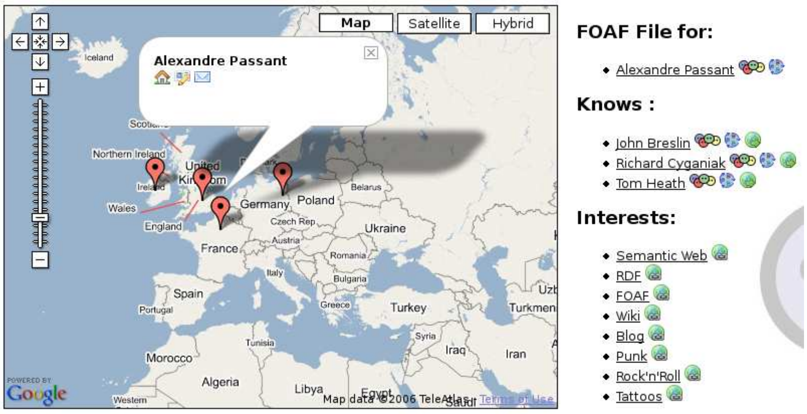
[2] `http://w3.org/2003/01/geo`

so that you can find people with the same interests as you[3]. It also provides an RSS 1.0 feed of the latest created and updated maps.

FOAFMap's source code strongly depends on PHOAF[4], a PHP5 API based on RAP[2] providing a set of classes and methods for easy information extraction from FOAF files without any knowledge of FOAF and RDF. FOAFMap and PHOAF handle FOAF and RELATIONSHIP[3] vocabularies to identify relations between people - RAP inference engine is used so that relations can be retrieved either they are defined by `foaf:knows` or `rel:xxx`. Finally, FOAFMap also uses MySQL to store some data as users and tags, and runs on LAMP.

3 Conclusion and Future Work

FOAFMap certainly won't evolve a lot in the future, as the goal was mainly to develop a basic prototype that could show connections between Semantic Web and user-oriented model of Web 2.0. Yet, I hope that such services can help end-users to see what FOAF and Semantic Web can bring us.



FOAF File for:

- Alexandre Passant

Knows :

- John Breslin
- Richard Cyganiak
- Tom Heath

Interests:

- Semantic Web
- RDF
- FOAF
- Wiki
- Blog
- Punk
- Rock'n'Roll
- Tattoos

Fig. 1. My personal FOAF profile towards FOAFMap

References

1. Brickley, D., Miller, L.: FOAF Vocabulary Specification. *Technical report, FOAF Project*, 2003. `http://xmlns.com/foaf/0.1`
2. Oldakowski, R., Bizer R., Westphal D.: RAP: RDF API for PHP. In *Scripting for the Semantic Web*, May 2005.
3. David, I., Vitiello, E.: RELATIONSHIP: A vocabulary for describing relationships between people `http://vocab.org/relationship/`

[3] `http://foafmap.net/tag/3`

[4] `http://gna.org/projects/phoaf`

A Semantic GIS Emergency Planning Interface Based on Google Maps[*]

Vlad Tanasescu and John Domingue

Knowledge Media Institute, The Open University, Walton Hall, Milton Keynes, UK
{v.tanasescu, j.b.domingue}@open.ac.uk

Abstract. We outline a generic graphical user interface for a Semantic Geographical Information System handling heterogeneous data sources through Semantic Web Services. This application aims to provide a goal oriented tool for emergency planners and decision makers of the Essex County Council. It uses Google Maps API, Firefox's implementation of JavaScript with E4X for XML handling, as well as AJAX techniques to access IRS-III Semantic Web Services execution platform. It allows access to resources relevant to the emergency context, as defined in an OCML ontology, and to communicate with relevant agents through BuddySpace's Instant Messaging services.

1 Application Description

In an emergency planning situation different agencies have to collaborate by sharing data as well as information. However, many emergency resources are not available online and interactions among agencies usually occur on a personal/phone/fax basis. The resulting system is therefore limited in scope and slow in response-time, contrary to the nature of the need for information access in an emergency situation.

To help alleviating this problem for the Essex County Council (ECC) emergency planning officers (EPO), we applied concepts based on the Semantic Geographical Information System (GIS) framework introduced in [4] to develop a prototype application. In the resulting system functionalities and relevant data sources are exposed by means of Web Services (WS), semantically described by OCML ontologies, and accessible to the user through a web interface using Google Maps. At the heart of the system stands IRS-III, a Semantic Web Services (SWS) execution platform described in [2]. For the time being it handles *accommodation* and *presence* related goal invocations, discovers and selects SWS that satisfy these goals, manages SWS orchestration and mediation, before executing WS provided by British Telecom. The mediated results are returned in a custom XML format, parsed by Firefox's implementation of JavaScript using the E4X idiom, and displayed on Google Maps.

BuddySpace [3] is used as a Jabber Instant Messaging protocol client to handle spatial presence; each user which specifies in his context a longitude and latitude,

[*] This work was supported by the DIP (Data, Information and Process Integration with Semantic Web Services) project. DIP (FP6 - 507483) is an Integrated Project funded under the European Unions IST programme.

possibly in a FOAF/RDF file, will appear on the map if relevant to the situation, as described in the emergency planning ontology.

2 Usage Example

A user defines a *snow hazard* or a *snow storm* (each offering different goals), before trying to contact relevant agents. The procedure is as follows:

1. Based on external emergency information the EPO draws a polygon on the map, then assigns a type of emergency to the region. Here, *a snow storm*.

2. Described in an ontology, the new instance has attached *features* and *goals*. Here three goals, one *gets shelters* at distance from the area, two others *connect* to BuddySpace and *get relevant presences*.

3. First, the user requests all *rest centres* inside the region, they are retrieved with their features and attached goals.

4. With that information the EPO *logs* into BuddySpace, then *contacts* the relevant persons to requests action or information.

A video of the interaction is available[1] as well as a live website[2] for testing with the last version of the Firefox browser. JavaScript code for the interface is available online, in '*.js' files.[3]

3 Conclusion and Future Work

This is work in progress. In the following versions, we plan to add meteorological information resources, which introduce other GIS concepts such as *fields*. Eventually our system should reach a level of functionality satisfying for the EPO as well as a test bed for further experiments in Semantic GIS.

References

1. Roman, D et al.: WSMO - Standard, WSMO Working Draft D2, 16 August 2004.
2. Domingue, J. et al.: IRS-III: A Platform and Infrastructure for Creating WSMO-based Semantic Web Services. Workshop on WSMO Implementations (WIW 2004), 2004.
3. Eisenstadt, M., Komzak, J. and Dzbor, M.: Instant messaging + maps = powerful collaboration tools for distance learning. Proc. TelEduc03, May 19-21, 2003.
4. Tanasescu, V. and Domingue, J., Toward User Oriented Semantic Geographical Information Systems, 2nd AKT Doctoral Symposium, 2006.

[1] http://irs-test.open.ac.uk/sgis-dev/vlad/sgis.htm

[2] http://irs-test.open.ac.uk/sgis-dev/

[3] *e.g.* http://irs-test.open.ac.uk/sgis-dev/spatial.js

Lego-Note: To Generate Semantic Web Content by Graphic Tagging

Jie Yang and Mitsuru Ishizuka

Department of Creative Infomatics
Tokyo University, Tokyo, Japan
yangj@mi.ci.i.u-tokyo.ac.jp, ishizuka@i.u-tokyo.ac.jp

Abstract. Lego-Note is an open source, browser-based semantic application inspired by folksonomy. Different from the other keyword-based, flat tagging systems, Lego-Note is featured by enriched graphic tagging based on a RDF model. The paper presents the implementation details of Lego-Note. The demonstration and source code can be found at https://sourceforge.net/projects/dom-sensus.

1 Introduction

Folksonomy like del.icio.us has proved a great success in recent years and gains more and more attention as a promising data source of the Semantic Web[1][2][3]. The Lego-Note presented in this paper attempts to extend the expression capability of folksonomy system, make it easier to obtain the semantic annotations, and thus to enhance the users experience further. To this end, Lego-Note is featured by:

- Tagging into the web page instead of the whole only
- Organizing tags in the form of labelled graphs
- Defining a RDF model of the graphic tagging[1].

2 Implementation

In this section, we describe the system functions and implementation details in terms of how a user might experience it.

The user goes to the web site of Lego-Note and starts browsing with the embedded browser. Right after the "GO" button is clicked, a piece of AJAX Extended [2] code is activated and sends the URL to a proxy server. The proxy server implemented in PHP then initials the request, and sends the returned page content back in the form of JavaScript Object Notation (JSON) . Here, the "cross domain" security restriction which prevents the user from tagging into the web content is evaded with the help of AJAX Extended.

[1] http://dom-sensus.sourceforge.net/

[2] http://ajaxextended.com/

After the returned content is rendered by the embedded browser, the user can either check the tag graphs made by the other users or add tags of himself. The tags take the form of the labelled graph. The node of the graph is a tag to the current page, which either points to the whole page, or to the selected content inside the page. The named edge which connects two nodes represents the user defined relation between two tags. An SVG based graphic editor which provides the basic graph manipulations such as adding, moving and deleting nodes and edges is implemented. The user can save his tag graph locally or share it publicly by saving to the Lego-Note server. The communication of saving and fetching the tag graphs is also implemented with AJAX. The server side functions are implemented with PHP and MySQL.

Untill now, the user can browse and tag any web page by entering a URL in the embedded browser. In order to preserve the user's browsing experience, link-following surfing is provided. This means when any link is clicked, the request should be captured and sent to the proxy server so that the next web page get loaded free of "the same domain restriction" as well. In Lego-Note this function is realized with JavaScript Behavior package[3].

Besides, to support the tag based browsing, a graph layout algorithm is provided by two JavaScript packages named graph.js[4] and prototype[5].

3 Discussion and Future Work

Although Lego-Note tries it best to be hardware and OS independent, the various web browsers (e.g. Firefox and IE) and SVG viewers make it a time-consuming work to support all of them. At present, Lego-Note only works with IE (above version 5.0) and Adobe SVG viewer 3.0.

The project is carried out under our observation and perspective on the folksonomy and Semantic Web. The target is to lower the barrier for the Semantic Web content production via the improved tagging system, and to provide us an infrastructure and data source for further researches like ontology mapping and negotiation. From the view of an application, Lego-Note is still an on-going project in its early stage, much more functions should be added.

References

1. Guy, M., and Tonkin, E: Folksonomies: Tidying up Tags? D-Lib Magazine, 12(1), (2006)
2. Mika, Peter: Ontologies Are Us: A Unified Model of Social Networks and Semantics. International Semantic Web Conference.(2005) 522–536
3. Tom Gruber: Ontology of Folksonomy:A Mash-up of Apples and Oranges. http://tomgruber.org/writing/ontology-of-folksonomy.htm

[3] http://bennolan.com/behaviour/

[4] http://aslakhellesoy.com/articles/2006/02/25/first-release-of-graph-js

[5] http://prototype.conio.net/