Enabling Semantic Web communities with DBin: an overview

Giovanni Tummarello, Christian Morbidoni, Michele Nucci

Dipartimento di Elettronica, Intelligenza Artificiale e Telecomunicazioni
Università Politecnica delle Marche, Via Brecce Bianche – 60131 Ancona (ITALY)
{g.tummarello,c.morbidoni}@deit.univpm.it,mik.nucci@gmail.com

Abstract. In this paper we give an overview of the DBin Semantic Web information manager. Then we describe how it enables users to create and experience the Semantic Web by exchanging RDF knowledge in P2P "topic" channels. Once sufficient information has been collected locally, rich and fast browsing of the Semantic Web becomes possible without generating external traffic or computational load. In this way each client builds and populates a 'personal semantic space' on which user defined rules, trust metrics and filtering can be freely applied. We also discuss issues such as end user interaction and the social aggregation model induced by this novel application.

1. Introduction

In this paper we present a novel kind of Semantic Web scenario which we call “Semantic Web Communities”. The idea is to enable end users to create and experience the Semantic Web by exchanging knowledge in P2P "topic" channels.

Such an application model can in a sense be though of as a file-sharing for metadata with on-top "community configurable" user interfaces (Brainlets). Similar to a file-sharing client, in fact, such application connects directly to other peers; instead of files, however, it downloads and shares RDF metadata about resources which the group has defined “of interest”. This creates a flow of RDF information which ultimately allows the participants to build rich personal Semantic Web databases therefore supporting high speed local browsing, searching, personalized filtering and processing of information.

In implementing this idea in our prototype of Rich Semantic Web Client (RSWC), DBin ([1]), it became immediately clear that a number of issues had to be resolved, relating to a great number of independent yet interconnected aspects.

For example, once data has been collected, the real issue becomes how to allow the user to interact with it in a natural way, e.g. in a way much more attractive and meaningful than a list of "properties" and "resources". While this "visualization problem" seems a separately treatable problem, we claim that in this scenario it is not. We propose, for example, to leverage the existence of "groups" by providing a way for a

"group leader" to suggest "interaction profiles" with the data that is exchanged within that context.

Upon joining a group, the user is then advised to download what we call a Brainlet, that is a package of configuration settings and a priori knowledge providing editing and browsing facilities to best interact with the information shared in the group.

2. Use 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 has selected the topic of interest and has connected to a semantic web P2P group, RDF annotations just start flowing in and out "piece by piece" in a scalable fashion. Such operations are clearly topic-agnostic, but for the sake of the demonstration let us take an example of possible use of DBin by a Semantic Web researcher.

For example, a user, let us call him Bob, who expresses interest in a particular topic and related papers (say "Semantic Web P2P"), will keep a DBin open (possibly minimized) connected with a related P2P knowledge exchange group. Bob will then be able to review, from time to time, new pieces of relevant "information" that DBin collects from other participants. Such information might be pure metadata annotations (e.g. "the deadline for on-topic conference X has been set to Y"), but also advanced annotations pointing at rich data posted on the web (e.g. pictures, documents, long texts, etc.). He could then reply or further annotate each of this incoming pieces of information either for his personal use or for public knowledge. If such replies include attachment data, DBin automatically takes care of the needed web publishing. At database level all this information is coherently stored as RDF. At the user level, however, the common operations and views are grouped in domain specific user interfaces, which are called "Brainlets".

3. Brainlets

Brainlets can be though of as "configuration packages" preparing DBin to operate on a specific domain (e.g. Wine lovers, Italian Opera fans etc..). Given that Brainlet include customized user interface, the user might perceive them as full "*domain applications run inside DBin*" which can be installed as plug-ins. In short Brainlets define settings for:

- The ontologies to be used for annotations in the domain
- A general GUI layout;. which components to visualize 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.
- Templates for readily available "pre cooked" domain queries.
- Templates for wizards which guide the user when inserting new domain elements (to avoid duplicated URIs etc)

- A suggested trust model and information filtering rules for the domain. e.g. Public keys of well known “founding members” or authorities,
- Basic RDF knowledge package for the domain

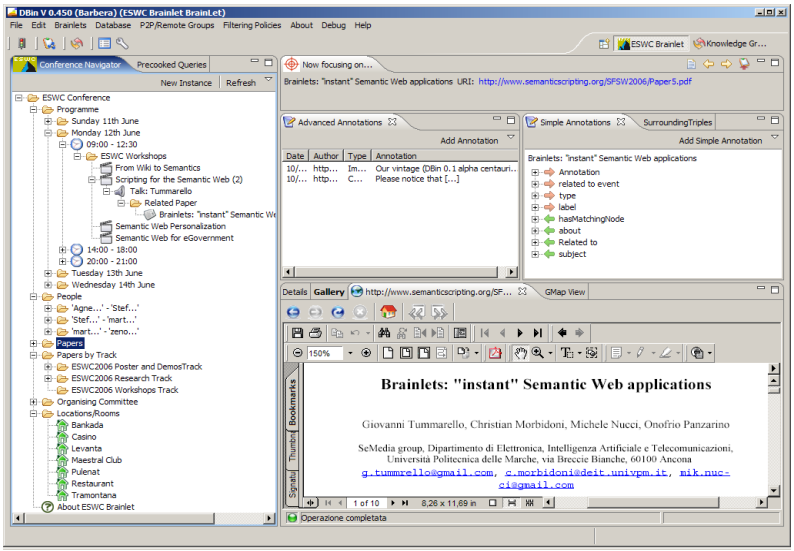

Figure 1: A screen shot of the ESWC 2006 Brainlet, dialing with presented papers, delegates and so on.

Creating Brainlets doesn't require programming skills, as it is just a matter of knowledge engineering (e.g. selecting the appropriate Ontologies) and editing of XML configuration files.
To create a new Brainlet, one copies from a given empty template which configures an eclipse plug-in to append a new "Brainlet" to the list of those known by DBin. This is done by means of an Eclipse RCP [2] extension point, which enables to install a plug-in with specified APIs and properties. Then, each Brainlet has its own XML configuration file, which, in addition to purely layout configuration (e.g. the positioning of the GUI blocks), allows one to define the Brainlet's core properties and facilities. The basic properties are the Brainlet name, version and URI, which usually indicates the web site from which to download the package. An overview of the other configurations and features follows.

3.1. Ontologies and default RDF knowledge

Probably the most important step in creating a new Brainlet is the choice of appropriate ontologies to represent the domain of interest. Once they have been identified, 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. In the same way basic knowledge of the domain can be included.

3.2. Navigation of resources

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 provided by the main DBin Navigator 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. The peculiarity of the approach is that the Brainlet creator can specify which is the 'relation' between each tree item and its children by associating them a semantic query. The results of such queries, which in DBin are expressed in SeRQL syntax [3], will be the item's children.

There can be multiple topic branches configured in the Navigator, specifying different kinds of relation between parent and child items. This enables the user to explore the resources of the domain under different points of view.

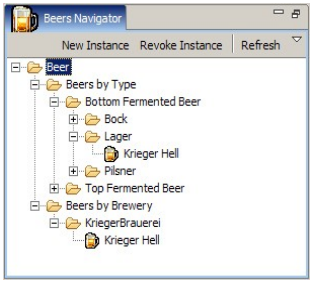

Figure 2: The navigator view can show different branches which organize concepts in different ways.

Figure 2 shows the Navigator view configured to display two branches, one (*beers by type*) gives an ontology driven hierarchical view on the domain, the other (*beers by brewery*) is a custom classification of the beers according the specific brewery producing them.

3.3. Selection flows

At user interface level, a Brainlet is composed by a set of 'view parts', as defined in the Eclipse platform terminology (Figure 1). Usually, each part takes a resource as a main "focus" and shows a 'something about' it (e.g. it's properties, images associated, etc...). Selection flows among these parts 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 done specifying, for each view part, which other ones will be notified when a resource has been selected.

3.4. “Precooked queries”

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.

3.5. URI wizards

It is very important to avoid different users to choose different URIs to indicate the very same concept. This can cause problems, for example, when two users have inserted the same beer while being off-line and later on they share their knowledge in the same P2P group. For this purpose we introduce the concept of URI Wizard, which defies procedures for assisting the user in the process of assigning identifiers to instances. Different procedures can be associated to different type of objects of the domain. For example, an intuitive procedure for choosing an identifier for Peroni beer might be to visit an authoritative web site (e.g. RateBeer.com), full-text search for 'Peroni' and then use the URL of the resulting web page.

Brainlets creators can choose among different preset procedures by XML configuration of various kinds of URI Wizards. Some of them, for example, creates a URI given a string that can represent the user nick, the time of creation, a well known name and so on, depending on the nature of the concepts (e.g. an MD5 for a file, the name a person, etc.).

This is a very simple methodology for choosing URI, and of course gives not absolute grantees, but we believe it to be very powerful and somehow sound, as it leverages the work of existing and established web communities.

3.6. Custom domain dependent annotation templates

Brainlets use the ontologies to assist the users in creating simple annotations (e.g suggesting which properties can be associated to a resource based on its type). A Brainlet creator can however also choose to create "complex annotation types" using an ad hoc OWL ontology. An example of such complex annotations is the "Beer Comparison", which directly compares beers, saying 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.

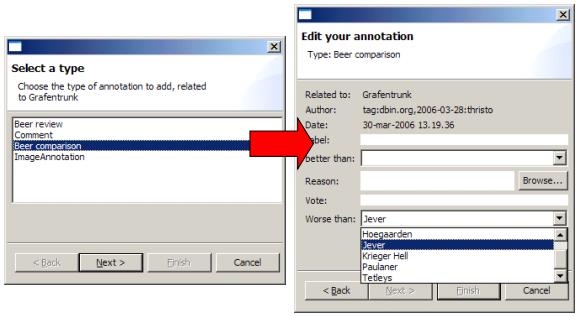

Figure 3: Advanced annotations are defined in OWL and auto generate property visualization and editing interfaces.

3.7. Ontology issue and social model

Brainlets are therefore preloaded by power users with domain specific user interaction facilities, as well as with domain ontologies suggested by the Brainlet creator. This seems to induce an interesting social model, mostly based on consensus upon Brainlets choice, which can help some of the well known issues in distributed metadata environments, a central one being the ontology mismatch problem. Brainlets, by providing an aggregation medium for ontologies, users and data representation structures, are

therefore good catalyst of the overall semantic interoperability process. As users gather around popular Brainlets for their topic of choice, the respective suggested ontologies and data representation practice will form an increasingly important reality. If someone decided to create a new Brainlet or Semantic Web application in general which could target the same user group as the said popular Brainlet, there would be an evident incentive in using compatible data structures and ontologies.

4. The RDFGrowth P2P engine: basic concepts

The RDFGrowth P2P algorithm, an early version of which is described in [4], constitutes the main channel by which a DBin client collects RDF data coming from other DBin users.

Previous P2P Semantic Web applications, such as [5] and [6], have explored interactions among groups of trusted and committed peers. In such systems peers rely on each other to forward query requests, collect and return results. In contrast, we consider the real world scenario of peers where cooperation is relatively frail. By this we mean that peers are certainly expected to provide some external service, but commitment should be minimal and in a "best effort" fashion.

The RDFGrowth algorithm has been designed to address this requirement of scalability and minimum commitment among peers, and is based on the peculiar philosophy of minimum external burden. Therefore peers are not required to perform any complex or time consuming operation such as query routing, replication, collecting and merging. As it is not in the scope of this paper to discuss in detail RDFGrowth, in this section we just give an overview of it.

As a design principle, given that a complex graph query could hog any machine's resources, we assumed that individual peers would not, in general, be willing to answer arbitrary external queries. In RDFGrowth any single peer would, if at all, answer just very basic ones, which are defined by the "RDF Neighbours" operator(RDFN).

The RDFN of a resource *a* within a graph *G*, is the closure on blank nodes of the triples of *G* surrounding *a*, and represents what is known about *a* at a given peer. This type of query is not only very fast to execute but can also be cached very effectively.

RDFGrowth uses the group metaphor to enable users to aggregate around topics of interest. When a user joins a particular group, DBin begins to collect and share only that kind of information which is of interest within the group. Such a topic's definition is given by the GUED(Group URI Exposing Definition), which is an operator capable of selecting from a triplestore a set of resources with common characteristics. A GUEDs can be implemented as simple sets of schema queries (e.g. all resources of type *ex:Paper* where *ex:topic* property value is "Semantic Web") and each group relays on one of them for advertising what can be shared within the community.

Once a group has been joined, the algorithm cycles over the set of 'on topic' resources locally known (the result of the GUED operator applied to the local DB) and, for each resource, searches other peers having new knowledge 'about' it. This is done by looking into a sort of Distributed Hash Table(DHT), in which each peer publishes the hashes of its RDFN (say simple MD5). When a certain condition on the RDFN hash is verified an actual metadata exchange is initiated with the other pee. In a simple version one can simply choose the first hash which is different from the local one, but more advanced methods can be used to try to guess the most profitable peer to ask to.

No “active information hunt” such as query routing, replication, collecting and merging is done. Such operations would require peers to do work on behalf of others that is again allowing peers to cause a potentially large external burden.

So, instead of querying around, in DBin a user browses only on a local and potentially very large metadata database, while the RDFGrowth algorithm “keeps it alive” by updating it in a sustainable, “best effort” fashion. A complete discussion is outside the scope of this introduction to the Demo, those interested can refer to [4] and other papers available from the DBin web site. As a result, keeping DBin open and connected to P2P groups with moderate traffic requires absolutely minimal network and computational resources.

5. URI Bridge Component

As we said before, the RDFGrowth algorithm enables the exchange of pure RDF metadata, so it is clear that some facility is needed in DBin to provide the user with the digital content referred by this metadata. Also an 'upload' mechanism is needed to allow users to be able to share his/her digital data (e.g. Images, text etc).

While the download mechanism is straightforward once a URL is available for a specific resource, as it can be retrieved, for example, over standard HTTP protocol, the uploading part requires some further considerations. The URIBridge is based on upload servers where users can store files they want to share (e.g. pictures, text, mp3s). After having uploaded a resource, the user is provided with a URL which can be used to create annotations about the data as well as to retrieve that data in order to visualize it.

DBin clients can be configured to work with one or more upload servers, much like an E-Mail client requires a SMTP server. While the default installation of DBin comes with a simple upload server, this limits the users to small files. For power users, installing a personal upload server is trivial, just deploying a simple PHP script.

6. Identities and authorship of annotations

In such a system, which deals with potentially large and unregulated communities, it is important to have information about who said what, in particular which user is the author of a particular annotation received from the network.

To do this we use a methodology based on the notion of MSG (Minimum Self Contained Graph), defined in [7] and enabling the decomposition of a graph in atomic units, and on the canonical serialization of RDF graphs suggested in [8].

This methodology enables RDF data to be signed at a fine granularity level and in an efficient way (MSGs are composed by more triples but it can be signed just by reifying a single one of them). It also assures that the context (in this case the authorship) will remain within the metadata when they will be exchanged over the network, as well as enable multiple signature to be attached to the same MSG, also at different times. As it is impossible to discuss in detail the digital signature process due to space limitation, please refer to [7].

When started up for the first time, DBin clients require the users to provide a valid URI which will act as an identifier for the user itself (for example a mailto URL or a

web page). Then a public and a private keys are generated; the private key is stored locally, while the public one is uploaded by means of the URIBridge, just as it happens for files. Every time a user will add an annotation to the system it will contain the user's identifier as well as the URL of the public key, and will be signed using the user's private key. In this way, after having received a piece of metadata from the P2P group, clients are able to retrieve the public key and to identify the author of the annotation, without caring about the provenance of the metadata itself.

Once the authorship of a MSG can be verified, a variety of filtering rules can be applied at will. These, in our system, are always non-destructive; information that does not match certain trust criteria can be hidden away but does not get deleted. It is straightforward, for example, to implement a local 'black list' policy, allowing users to add authors to that list and to filter the local knowledge in order to hide all the information signed by the same user's identity.

7. Conclusions and related works

Aspects of DBin can be compared with [9], [10] and [11]. Details of this comparison are not possible due to lack of space, but DBin stands out as a end user centered application which provides an all round and integrated Semantic Web experience. By this we mean that, albeit in perfectible forms, DBin provides a single interface that entirely cover the needs of complex use cases. Users can browse, query, publish and cooperatively create Semantic Web databases. Media inserts can be seen together with the relative metadata, incoming knowledge can be filtered based on local policies and information locally produced is digitally signed. More than this, users are given the ability to create “Semantic Web Communities” by creating both application environments, Brainlets, and exchange and meeting places, RDFGrowth channels. DBin is an Open Source project (GPL). Further documentation and compiled executables can be downloaded at *http://dbin.org*, where a few minutes screen demo is also available.

References

[1] "The DBin project" http://www.dbin.org
[2] "Eclipse Rich Client Platform", http://www.eclipse.org/rcp
[3] J. Broekstra, A. Kampman , "SeRQL: An RDF Query and Transformation Language", ISWC, 2004
[4] G. Tummarello, C. Morbidoni, J. Petersson, P. Puliti, F. Piazza, "RDFGrowth, a P2P annotation exchange algorithm for scalable Semantic Web applications", P2PKM Workshop, 2004
[5] W. Nejdl, B. Wolf , "EDUTELLA: A P2P Networking Infrastructure Based on RDF", WWW, 2002
[6] P.A. Chirita, S. Idreos, M. Koubarakis, W. Nejdl, "Publish/Subscribe for RDF-based P2P Networks", ESWS, 2004
[7] G. Tummarello, C. Morbidoni, P. Puliti, F. Piazza , "Signing individual fragments of an RDF graph", WWW, 2005
[8] J. Carroll, "Signing RDF Graphs", ISWC, 2003
[9] D. Huynh, S. Mazzocchi, D. Karger , "Piggy Bank: Experience the Semantic Web Inside Your Web Browser",ISWC, 2005
[10] J. Broekstra, M. Ehrig, P. Haase, F. van Harmelen, M. Menken, P. Mika, B. Schnizler, R Siebes , "Bibster - A Semantics-Based Bibliographic Peer-to-Peer System." SemPGrid Workshop, 2004
[11] D. Quan, D. Karger, "How to Make a Semantic Web Browser", WWW , 2004