Distributed creation and revision of domain knowledge based on Rich Semantic Web Clients

Giovanni Tummarello, Christian Morbidoni, Michele Nucci

Dipartimento di Elettronica, Intelligenza Artificiale e Telecomunicazioni
Università Politecnica delle Marche,
Via Brecce Bianche – 60131 Ancona (ITALY)

Abstract

In this paper we present a novel kind of personal application: Rich Personal Semantic Web Clients. The idea is to enable to user to create and experience the Semantic Web as a collection of local databases which interoperate in semantic P2P "topic" channels by exchanging patches of RDF information. Once sufficient information has been collected locally, rich, fast browsing of the "Semantic Web" becomes possible without generating external traffic or computational load. Also user defined rules, trust metrics, filtering, can be freely applied. "Semantic Web" data can be furthermore integrated with personal local semantic data such as local sources or Desktop information. A prototype of such client, DBin, is presented and then we describe issues such as user interfaces and the social aggregation model induced by this novel application model.

Introduction

In this paper we present a novel kind of personal application, Rich Personal Semantic Web Clients (RPSWC). The idea is to enable to user to create and experience the Semantic Web as a collection of local databases which interoperate in semantic P2P "topic" channels by exchanging patches of RDF information.
Such application model can in a sense be though as a file-sharing for metadata with on top "community configurable" user interfaces. 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 DBin [1] it became immediately clear how a number of issues had to be resolved, relating a great number of independent yet interconnected aspects.
For example, once data has been collected, the real issue becomes how to allow the user 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 suggested to download what we call a Brainlet, that is a package of configuration and a priori knowledge to best interact with the information shared in the group.

A novel scenario

The scenario considered in this paper is new under many aspects. Many of the P2P approaches based on Semantic Web technologies proposed so far [2][3][4][5], use metadata and ontologies to build a semantically structured definition of the resources a user is searching for and/or is offering to other participants in the network. In general these system are finalized to manage and optimize the retrieval of actual files, like textual documents, audio, video and so on. In such systems a user who formulates queries like "All publications about Semantic Web by author X", knows in advance precisely what to search for, thus having a certain knowledge of the domain.
The use cases we take in consideration are more like that of a user, perhaps new to Semantic Web, how is interested in learning (more) about the research that is going on. He/she is supposed simply to join a 'Semantic Web' topic group, to receive new and unexpected information, for example papers, the conferences on which they have been published, their author's names. Users are not really interested exclusively in "hits" locating remote resources, but rather into learning as much as possible about them so that more uses of this information become possible (e.g. Personalized browsing, joining with local information etc).

The RDFGrowth P2P engine: basic concepts

In this section we shortly describe the P2P algorithm which is the core module of DBin, our prototype implementation.
Previous P2P Semantic Web applications, such as [2][5], have explored interactions among groups of trusted and committed peers. In such systems peers rely on each other to forward query requests and collecting and returning 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 peer are not required to perform any complex or time consuming operation, such as query routing, replication, collecting and merging.

RDFN: the only query allowed

As a complex graph query might simply hog any machine, the only RDF query allowed during metadata exchanges, is a simple and basic one, which not only is fast to execute but also can be cached very effectively. It is defined by the RDFN (RDF Neighbors) operator, which retrieves meaningful peace of information, in the form of RDF metadata, directly connected to a specific resource.
In detail the RDFN (RDF Neighbours) of a URI is a graph composed roughly by all the triples that have it as subject or object. In case some of these also refer to blank nodes, then the RDN is also recursively composed by all the triples until just "ground" (URI or Literals) node form the "edge" of the RDFN. Figure 1 shows the RDFN of a sample resource and its composing MSGs.

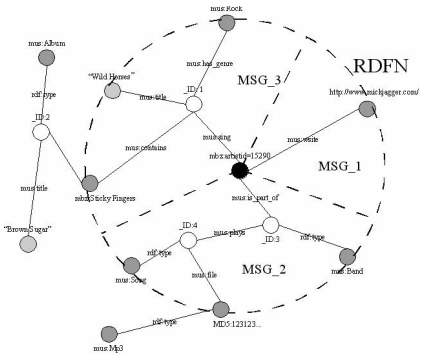

Figure 1 *The RDFN of the resource painted in black is delimited by the dotted circle. White nodes represent blank nodes. The RDFN is composed by several slices, each one is an MSG, basically a closure on blank nodes starting from a given triple, and represents the minimum unit of knowledge that is exchanged.*

MSGs

The RDFN(a) can be also considered as the union of all the MSG which involve the resource *a*. An MSG is a subgraph with a well defined structure and is actually the minimum amount of information that can be exchanged in the system. See [6] for details about the RDFN and MSG definitions and theory. Its interesting properties allows to exchange information in a fine granularity, incremental fashion, along with its context. We will see later, in fact, how authorship information can be efficiently attached at MSG level, allowing personalized trust policies and information revocation.

GUED (Group URIs Exposing Definition)

Users connects to a RDFGrowth network by selecting a topic group and joining it. The client then receive an operator, which defines, and is able to retrieve from a graph, the resources which the group is about. A GUED can be implemented as a set of queries. As an example, for a Michael Jackson group, a possible GUED might be "select all the URIs identifying his songs, albums, concerts and interviews", in respect to an agreed ontology.
Once received the GUED operator, the peer execute it once on his DB and the resulting set of URIs are "published" in the p2p network, as an advertisement that they are in fact of interest and will be willing to answer requests from other peers about the "RDF Neighbours" (RDFN, see next section).

RDFN Hashes and exchange strategy

The algorithm cycles over the set of 'on topic' URIs (selected by the GUED) and for each of them searches for peers who have different surrounding informations (RDFN) than the local ones. This process is performed by looking into a sort of DHT in which hashes of RDFNs (say simple MD5) are exposed by each peer. Once an hash is detected which is different from the one exposed by the local peer, an exchange is initiated.
During the exchange peers synchronizes their knowledge about the resource. In addition to simple hashes more advanced heuristics cab be applied to identify new information present in the network and choose the peer with who it is more profitable to ask to.

Considerations

A key point in this approach to metadata sharing is that the algorithm grows a local triple store at every peer, this not only enables fast browsing and complex query execution (performed using local computational power on the local DB, no external commitment), but also make possible for metadata to naturally cross the borders across communities. As an example, suppose that in a “movie community” someone posts a picture of an actor and in a “rap music” community the same actor has been mentioned as performer. Then a user participating at same time to both communities would, by the logic of the RDFGrowth algorithm, make so that the picture is also “posted” in the movie group.
While the approach fits well the target scenario, there are some shortcoming that might be mentioned:

- As the database continuously expands a large HD space might be required to store all the metadata from numerous or big communities. Given the cost/capacity of modern drives, we think that the HD space problem is very secondary. Actual content caching policy e.g. Images, can be changed at will, see the section about the URIBridge.
- Although the time it takes to get a new piece of information is bounded, the approach is not suitable for real time structured information (e.g. Current weather in RDF)
- It takes some startup time for a new user to get enough

information for a meaningful navigation and querying.

Such a “growth only” scenario matches the monotonic nature of the RDF semantics. To obtain more information about a resource can’t in fact “hurt” since, by definition, previously inferred statements will still hold when new data becomes available. It is of course possible, in the real world applications, to rely on “context” information to later apply non monotonic rules on the local database, but it should remain a local peer decision to do so with no consequences on the shared knowledge. Digital signatures are an example of such context information which support several fundamental higher level non monotonic behaviours of the overall system.

Dealing with the actual data

Authorship rather than provenance

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.
As the system is based on high replication of metadata over the P2P group, to look at the provenance of the metadata (which peer sent it to me), is useless in order to obtain authorship information. In fact, if Bob sends a comment about a paper to someone else, it does not mean that Bob himself wrote the comment, as it might come from an other user, who again might obtained it by someone else.
As is evident that tracking the provenance is not thinkable in our system, we have to provide a methodology for 'marking' every peace of metadata added to the system with verifiable statements about the authorship.

User identities and digital signatures in DBin

The MSG definition and properties highlighted in the previous section, when combined with a canonicalized serialization as suggested in [7], enable signing MSGs themselves in a efficient way: attaching the signature information only to a single triple composing the MSG. This methodology, described in detail in [8], 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 enables multiple signature to be attached to the same MSG, also at different times.
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 secret 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, that is a certain number of MSGs, the annotation itself 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 peace 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.

Information filtering and revision

The mechanism described in the previous section provide a framework for authorship assessment, trust based filtering and information revision.
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 doesn't match certain trust criteria can be hidden away but does not get deleted. Is it 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.
Moreover, the hash of a signed MSG, can be used as an Inverse Functional Property (IFP, see), that is, as a unique way to name to the MSG itself. This in turn can be used in a subsequent MSG to indicate the one that it substitutes. Given that the paternity of this subsequent MSG can be verified to be identical, the client can safely perform the information update, no matter where it received the update patch from.
DBin also includes a preliminary support for a centralized certification authority, dubbed RDFTrust. By obtaining a signed certificate (which involves a time consuming identification procedure), users can enjoy instantly an higher degree of trust. If trust was to be abused (e.g. By spamming or malicious “ontology violations”) the certificate would be revoked thus causing all the previously inserted information to “sink” in each DBin installation. The certification authority web site is currently being implemented as a web application with a matching application API.

User interface: “Brainlets”

There has been a lot of work recently on Semantic Web visualization and a number of user interface have been proposed [9], [10], [11], [12], [13]. While pro and cons can be argued for each specific approach, it is clear that user interface issues are a complex issue with no clear single solution.
In designing the architecture of a RPSWC, rather than a single answer to this issue, we thought about a general set of “application oriented” generic GUI tools by which 'power users' can build applications specifically targeted to the domain of interest.

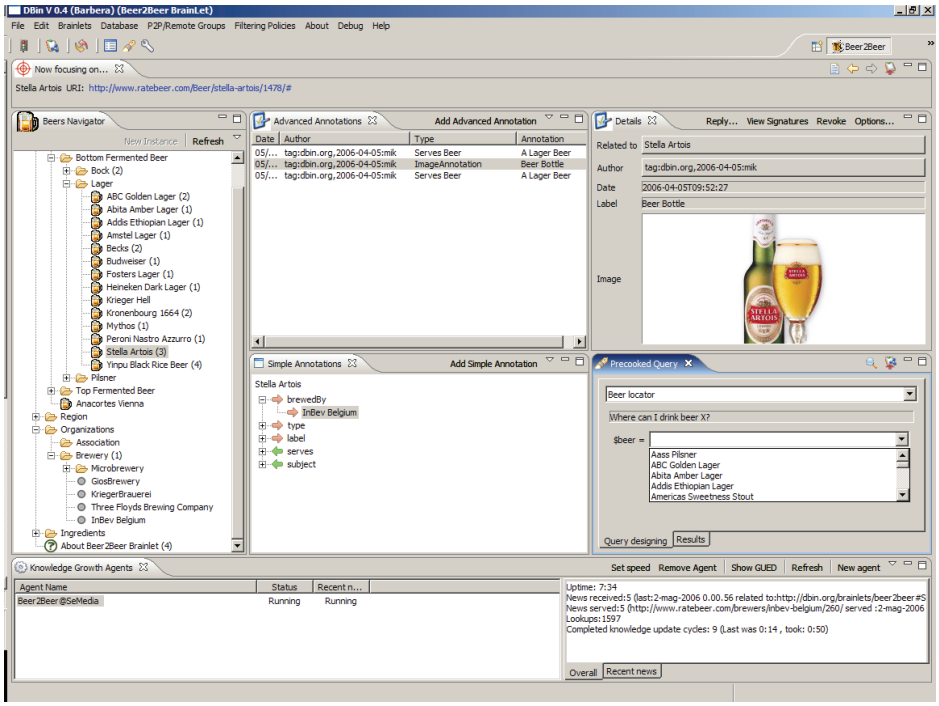

Figure 2 *A screen shot of the Beer2Beer Brainlet running. The principal "views" are: an ontology (and instances) browsing Navigator, the Knowledge Agents view, showing statistics about the currently running knowledge agents, and a set of "Annotation" views. Among these a comment view, a picture gallery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

In DBin these *domain specific applications*, called "Brainlets", are implemented as plug-in modules. Brainlets can be though as "configuration packages" preparing the client to operate on a specific domain (e.g. Wine lovers, Italian Opera fans etc..). Given that Brainlets include customized user interface, the user might perceive Brainlets as full "domain applications" which are run by the RPSWC. Figure 2 shows a screen shot of "Beer2Beer", our first XML based Brainlet.

Domain knowledge driven configuration

Each specific domain context has an associated 'view of the world', which is not only the domain concepts and their taxonomy, but can also be thought in terms of peculiar paths that domain experts and users follow to find relevant informations or to relate and browse objects.

From a user interface point of view this means that in some cases users might be likely see an explicit relation between a concept and its sub-concepts (e.g. Beer related to specific kins of beers), while in others it would be better to think about more specific and domain dependent relations (e.g. jump from a Beer to the pubs located in London which serve it).

The knowledge of these 'frequently used paths' to relate information within a domain are owned by domain experts. The idea in the Brainlet paradigm is to allow power users to use this knowledge in configuring the domain interface. This includes controlling how concepts and instances are presented to the end user, providing useful editing and browsing facilities and assisting the user in creating and managing objects of the domain.

For the this model to be applicable in practice, it is crucial that Brainlet should be created ad configured as much as possible with no programming skills. In DBin implementation, basic Brainlets can be configured by editing XML files and or OWL ontologies and more advanced ones can however be made including custom Eclipse plug-ins as needed.

In short a Brainlet is composed by :

- The setup information for the RDFGrowth algorithm and the transport layer to connect with others using the same Brainlet (namely, at this point, the name of the rendezvous servers, the channels and the GUED)
- 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”.- Support material, customized icons, help files etc..
- A basic RDF, GUED conforming, knowledge package

In the next sessions we will refer to some of the 'views' composing the interface shown in Figure 2 and discuss what they represent and how, properly configured by a domain expert, they can make Brainlets powerful tools to experience Semantic Web communities and contribute to build common domain knowledge.

The main domain navigator

The main Brainlet navigator (left view named 'Beer Navigator', in Figure 2) shows a precise organization of the domain concepts and serve as a starting point for the user to browse the underling RDF metadata.
As, graph based visualizers are notably problematic when dealing with a relevant number of resources, in the navigator we adopted a solution 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 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 expressed using the SeRQL syntax [14]). 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:

Each Topic has a URI associated (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:

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.

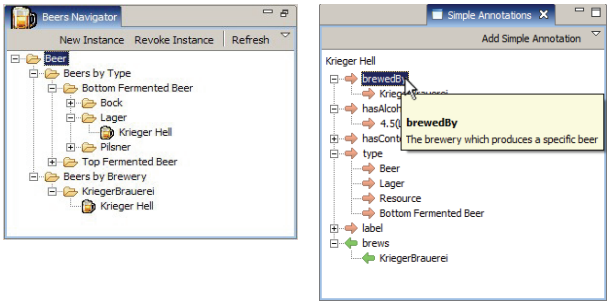

Figure 3: *a) The simple annotation view shows the properties of the selected resource (left figure). b) A navigator view generated by the configuration shown above. The instance of a beer can be reached following different paths, by type or by brewery(right figure).*

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:

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 an action insider the view by the user will cause the manager to change its selected URI. As many "Selection manager" can be created as necessary to support to information flow logic withing 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:

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.

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.

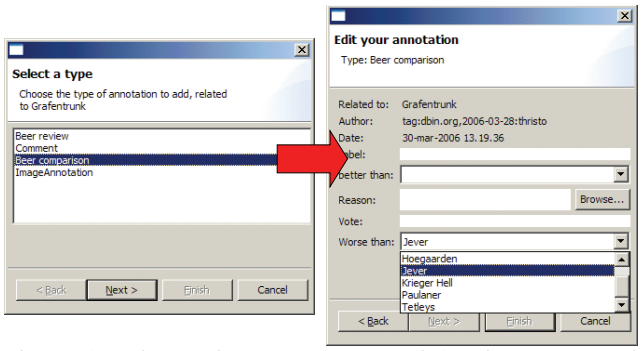

Figure 4: *Advanced annotations are defined in OWL and auto generate property visualization and editing interfaces.*

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 in fact 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 .

An XML example configuration is given:

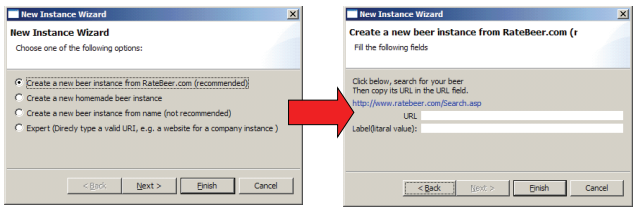

Figure 5: *Choosing to insert a new instance of beer the user is prompted with a wizard and can choose different procedures to create a URI.*

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, data representation structures, are therefore good catalyst of the overall semantic interoperability process.

In fact, 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, it is clear that there would be incentive in using identical or somehow compatible data structures and ontologies.

Software Engineering/License

The prototype we discussed here, DBin, is programmed in Java and based on the Eclipse Rich Client platform. As such, DBin is naturally multi-platform, features an OS native look and feel and is highly extensible trough the well known Eclipse plug-ins and extension points technology. Both the framework and modules presented here are open source. Licensing terms have not been settled yet, but they're expected to be either LGPL, BSD or CPL.

Conclusions

In this paper we presented a novel Semantic Web scenario where information is created, exchanged and browsed by Rich Personal Semantic Web Clients. In doing so we illustrated DBin, our prototype meant to demonstrate the usefulness and scalability of such model.

While such model does not allow the user to immediately perform queries or browsing, we believe that this is in fact a familiar paradigm for Internet users as it is not so much different from popular P2P file sharing applications.

In the same way as many users have gotten used to wait to obtain data by running a classic P2P file sharing, DBin users will “peacefully” discover new information about topics in which they express interest in.

In a possible scenario, a user is interested about a very specific music author and leaves DBin running in a channel about it. From time to time he learns of pictures taken by fans, comments, reviews, etc..

Content and annotations produced by the user, on the other hand, can reach precisely those who had expressed interest in it and naturally cross the boundary of the P2P group they were posted originally to. Given RDFGrowth design in fact, relevant annotations are intrinsically and automatically bridged by the peers that visit multiple groups or return at later times.

While there are multiple solutions inside DBin that might be interesting per se, we believe the most important aspect is the holistic integration of the components under a single "scenario philosophy", in other words, the ability for such application to enable real Internet users, for the first time, to "take a look" from the top of the Semantic Web tower.

To enable this, we propose pragmatic solutions that suggested by the scenario itself. For example, we leverage the existence of group and their "maintainer" enabling them to provide Brainlets. These, in turn, transparently provide the users with pragmatic answers to notable Semantic Web open research issues such as trust, ontology management, user interface, etc...

While such solutions can hardly be thought as satisfying in to all possible user needs, we believe that they might in fact be "good enough" for a large number of use cases and user interest groups.

References

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

[2] Wolfgang Nejdl, Boris Wolf, EDUTELLA: A P2P Networking Infrastructure Based on RDF, WWW2002, Honolulu, 2002

[3] Min Cai, Martin Frank, RDFPeers: A Scalable Distributed RDF Repository based on A Structured Peer-to-Peer Network, 13th International World Wide Web Conference WWW2004, New York, 2004

[4] Wolfgang Nejdl, Wolf Siberski, Martin Wolpers, Alexander L”ser, Ingo Bruckhorst, SuperPeer Based Routing and Clustering Strategies for RDF Based Peer-To-Peer Networks, Twelfth International WWW03 Conference, Budapest, 2003

[5] Paul - Alexandru Chirita, Stratos Idreos, Manolis Koubarakis, and Wolfgang Nejdl , Publish/Subscribe for RDF-based P2P Networks, ESWS, Heraklian, Greece, 2004

[6] Giovanni Tummarello, Christian Morbidoni, Joackin Petersson, Paolo Puliti, Francesco Piazza, RDFGrowth, a P2P annotation exchange algorithm for scalable Semantic Web applications, First P2PKM Workshop, Boston , 2004
[7] Jeremy Carroll, Signing RDF Graphs, ISWC2003 , 2003
[8] G. Tummarello, C. Morbidoni, P. Puliti, F. Piazza, Signing individual fragments of an RDF graph, WWW2005, poster track, 2005
[9] R. Albertoni, A. Bertone, M. De Martino, Semantic Web and Information Visualization, Proceedings of the First Italian Workshop on Semantic Web Applications and Perspectives, Ancona (ITALY), 2004
[10] , RDF Gravity - RDF Graph Visualization Tool, Technical Report: HPL-2004-57 ,
[11] E Pietriga , Isaviz: a visual environment for browsing and authoring rdf models , 11th International World Wide Web Conference , 2002
[12] RDFX, Technical Report: HPL-2004-57 ,
S. Mazzocchi, P. CiccareseWelkin, a graph-based RDF visualizer2004http://simile.mit.edu/welkin/
[14] Jeen Broekstra, Arjohn Kampman, SeRQL: An RDF Query and Transformation Language, ISWC 2004, Hiroshima, Japan, 2004
[15] Leo Sauermann, Gnowsis Semantic Desktop ISWC2004 Demo, ISWC 2004 - Demo Track, 2004
[16] Robert MacGregor, Sameer Maggon, Baoshi Yan, MetaDesk: A Semantic Web Desktop Manager, International Workshop on Knowledge Markup and Semantic Annotation (SemAnnot 2004), Hiroshima, 2004