A Metadata Model for Semantics-Based Peer-to-Peer Systems

Jeen Broekstra[13], Marc Ehrig[2], Peter Haase[2], Frank van Harmelen[1], Arjohn Kampman[3], Marta Sabou[1], Ronny Siebes[1], Steffen Staab[2], Heiner Stuckenschmidt[1] Christoph Tempich[2]

[1]Vrije Universiteit Amsterdam
{jbroeks, frankh, marta, ronny, heiner}@cs.vu.nl
[2]Institute AIFB, University of Karlsruhe, D-76128 Karlsruhe
{ehrig,staab,tempich}@aifb.uni-karlsruhe.de
[3]AIdministrator BV, Ammersfoort
{jeen.broekstra, arjohn.kampman}@aidministrator.nl

Abstract. Peer-to-Peer systems are a new paradigm for information sharing and some systems have successfully been deployed. It has been argued that current Peer-to-Peer systems suffer from the lack of semantics. The SWAP project (Semantic Web and Peer-to-Peer) aims at overcoming this problem by combining the Peer-to-Peer paradigm with Semantic Web technologies. In the course of our investigations it turned out that the nature of Peer-to-Peer systems requires some compromises with respect to the use of semantic knowledge models. In particular, the notion of ontology does not really apply as we often do not find a *shared* understanding of the domain. In this paper, we propose a data model for encoding semantic information that combines features of ontology (concept hierarchies, relational structures) with a flexible description and ranking model that allows us to handle heterogeneous and even contradictory views on the domain of interest. We discuss the role of this model in the SWAP environment and describe the model as well as its creation and access.

1 Motivation

The essence of Peer-to-Peer (P2P) is that nodes in the network directly exploit resources present at other nodes of the network without intervention of any central server. The tremendous success of networks like Napster and Gnutella, and of highly visible industry initiatives such as Sun's JXTA, as well as the Peer-to-Peer Working Group including HP, IBM and Intel, have shown that the P2P paradigm is a particularly powerful one when it comes to sharing files over the Internet without any central repository, without centralized administration, and with file delivery dedicated solely to user needs in a robust, scalable manner. At the same time, today's P2P solutions support only limited update, search and retrieval functionality, e.g. search in Napster is restricted to string matches involving just two fields: "artist" and "track". These flaws however make current P2P systems unsuitable for knowledge sharing purposes.

Metadata plays a central role in the effort of providing search techniques that go beyond string matching. Ontology-based metadata facilitates the access to domain knowledge[1]. Furthermore, it enables the construction of exact queries. Existing approaches of ontology-based information access almost always assume a setting where information providers share an ontology that is used to access the information[2]. In a Peer-to-Peer setting, this assumption does not longer hold. We rather face the situation, where individual peers maintain their own view of the domain in terms of the organization of the local file system and other information sources. Enforcing the use of a global ontology in such a setting would mean to give up the benefits of the Peer-to-Peer approach mentioned above. Therefore, one has to find a way to deal with the existence of multiple, distributed and frequently changing views on the domain.

In this paper, we propose a metadata model that combines ontological structures with information needed to align, evolve and use these for query processing. In section 2, we explain the requirements for the metadata model that guided its development in more detail. The third section focuses on the SWAP environment in which the metadata model is used. The model itself is introduced in section 4. In section 5 we describe the methods supporting the creation, use and update of the metadata model. We conclude with a discussion of open problems.

2 Requirements

We will illustrate the requirements for the proposed system with the short scenario described in the following. Virtual organizations or large companies impose a complex situation, with respect to number of domains, conceptualizations and documents onto a peer-system for knowledge sharing. Typically their organizational units are distributed according to expertise or organizational tasks, such as development and marketing. A small case study of a virtual organization in the tourism domain is used as real world example. The virtual organization comprises public authorities, hotels and event organizers. The public authorities require the number of guests visiting the country to plan for example public transport and waste management. Event organizers can customize their offerings according to the number of visitors and the age. Hotels can publish this information to make the stay more pleasant. Today the exchange of this kind of information is time consuming, not in time and error prone, although it is often available in electronic form at every level. However, the different organization have diverse objectives and therefore use different conceptualizations of their domains.

From a technical point of view the different organizations can be seen as one or many independently operating nodes within a “knowledge” network. Nodes can join or disconnect from the network at any moment and can live or act independently of the behavior of other nodes in the system. A node may perform several tasks. Most important is that it acts as a peer in the network, so it can communicate with other nodes to achieve its goals. But apart from that it may act as an interface to interact with the human user of the network, or it may access knowledge sources to accomplish its tasks. One node may have one or more knowledge sources associated with it. These sources contain the information that a peer can make available to other peers. Examples are a user's filesystem and mail folders or a locally installed database. A node must be designed to meet the following requirements that arise from the task of sharing information from the external sources with other peers:

– Multiple sources of information
– Mostly uniform treatment of internal and external sources
– Multiple views on available information
– Support for query answering and routing
– Distribution of information within the network

The metadata model needs to reflect these requirements. We derive some main objectives for the metadata model with emphasis on the information mediation.

Integration Each piece of knowledge requires metadata about its origin. To retrieve external information, the metadata needs to capture information about where the piece of information was obtained from. This information will allow to identify a peer and locate resources in its repositories.

Information heterogeneity As information is added from a variety of peers, inconsistencies may occur in a local repository.

Information needs to be assigned a confidence rating, such that the system will be able to handle heterogeneity and provide useful information. Similarly, a level of trust can be assigned to peers to model their reliability.

Furthermore, as each peer uses its own local ontology, mappings may be required to overcome the heterogenous labelling of the same objects.

Security Some information may be of private nature and should not be visible to other peers. Other information may be restricted to a specific set of peers. The metadata model needs to provide means to express these security policies.

Caching Within Peer-to-Peer systems the availability of other peers is not always guaranteed. Moreover, some peers may have better connectivity, in terms of bandwidth, to the rest of the network than other peers. Hence, to improve network efficiency, caching of information can be useful. The caching mechanisms needs to be transparent to the user, but must be captured by the metadata model.

3 The SWAP environment

The SWAP environment[1] is a general infrastructure which was designed to meet the requirements on a knowledge node . The proposed architecture consists of three representational components: (1) knowledge sources (lower left corner of figure 1), (2) individual views on these information sources generated by the user interface (upper right corner of figure 1) and (3) and local node repository (upper left corner of figure 1). Together with additional information provided by other peers in the network these sources make up the knowledge available to a peer. The overall architecture of a SWAP node as shown in the figure is described in [3].

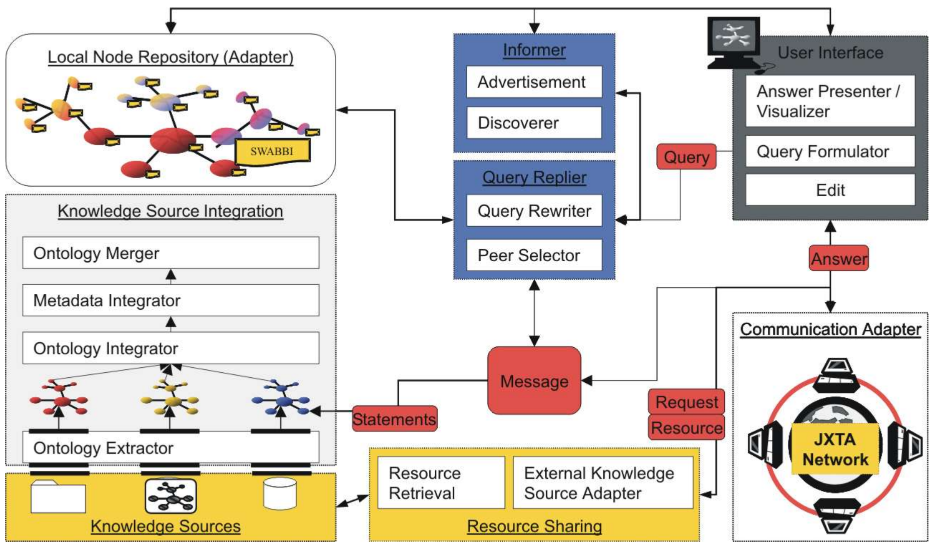


Fig. 1. Abstract Architecture of a SWAP Node

Internal Knowledge Sources: Peers may have local sources of information such as the local file system, e-mail directories or bookmark lists. These local information sources represent the peer's body of knowledge as well as his basic vocabulary. These sources of information are the place where a peer can physically store information (documents, web pages) to be shared on the network.

Individual Views: Peers provide different views on the information they have in their local sources as well as onto information on the network. The views can be implemented using different visualization techniques (topic hierarchies, thematic

[1] http://swap.semanticweb.org

maps, etc). There are some predefined views that correspond one-to-one to the different sources of information. Additionally, the user can define more complex views that range over different sources of information.

Local Node Repository: In order to manage the different information models and views as well as information acquired from the network, each peer maintains an internal working model. This model provides the following functionality:

– Mediate between views and stored information
– Support query formulation and processing
– Specify the peer's interface to the network
– Provide the basis for peer ranking and selection

The working model is not an ontology in the classical sense. It is rather a knowledge structure, which provides an integrated model of the different structures that coexist in the network and the peer itself. The different structures are stored in a single model, individual elements of the model are annotated with their source and a ranking representing the peer's belief in their plausibility. The complete model may contain inconsistencies, in order to receive a consistent model a part of the complete one has to be chosen on the basis of the plausibility ranking.

4 The SWAP Metadata Model

The SWAP environment aims at providing a general view on the knowledge each peer has. It should facilitate the access to different information sources and enable the user to take advantage from other peers' knowledge. Therefore a metadata model was designed which provides semantics to annotate external as well as internal data. Information from different information sources and from other peers can be integrated with this metadata model to enable a later retrieval of the underlying information items. If the information source is a file system, the information item, viz. file, must be retrievable. Furthermore, it allows to cache information to make the entire network work more efficiently. Another purpose of the metadata model is to deal with the information heterogeneity which is inherent in Peer-to-Peer systems.

4.1 Detailed description

As a response to the objectives, we define a SWAP specific metadata model in RDF(S)[4]. An overview of the model is given in figure 2. The complete definition of the model is available at:

`http://swap.semanticweb.org/2003/01/swap-peer#`

The model consists of two RDFS classes namely the "Swabbi"-class and a "Peer"-class. Each of these objects is augmented by several properties which allow for the above described objectives.

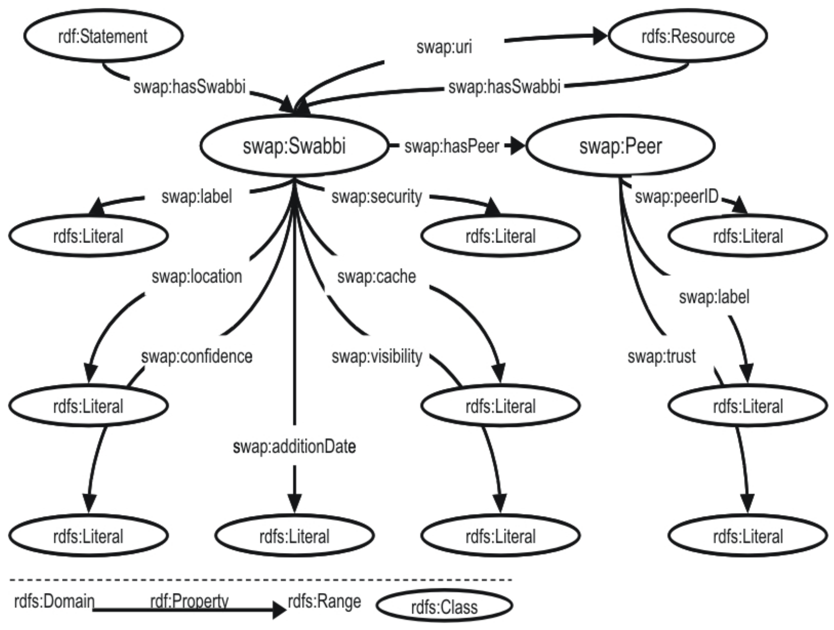


Fig. 2. The SWAP metadata model

Swabbi: Each piece of content information links to a “Swabbi”-object. This object contains the meta-information or links to it.

Peer: For each information we have to save from which peer it originates from. Therefore the local repository stores different information about each known peer. The information is grouped in the Peer object. The “Swabbi”-object links to the corresponding peer.

peerID: This is the first attribute stored within the peer object. Each peer has a unique ID to be identified. For our purposes this will be the *JXTA UID*.

peerLabel: The second peer attribute gives the peer label, which is a human readable and understandable description of the peer. Natural text is used for this. An example could be: “Marc Ehrig Notebook”.

peerTrust: The last peer attribute is a measure to include trust. Some peers might be more reliable than others. The faith in responses of peers changes with every message containing true or false information. It can be defined manually for specific peers. To control the peerTrust it is defined to have a value between 0 and 1, with zero meaning having no faith at all and one the peer being very trust-worthy.

uri: Each piece of information was originally created on one peer. To keep track of the origin of the information and its primary URI this is explicitly saved among the metadata. With this information it is possible to unambiguously address an object across the network. The URI-attribute is also required for mapping which will be explained at another point in this document.

location: Whereas the URI is an identifier within the ontology of the local repository, the location-attribute is a physical identifier. If we want to access a document we do not only need the document-object in the ontology but also the address where the document can be accessed e.g. file://c:/Projects/myfile.txt. Only the peer where the information is physically stored needs to be able to interpret the expression. The location information is also required when doing updates of the local repository.

label: The label saves how the specific information is called on the peer it originates from. The label-attribute is formulated in natural language, so users can also understand it (e.g. bank, financial institution). While URIs are only machine readable the label is human readable, but the meaning and uses are equivalent. As one concept can have different names on different peers, this property is added to each "Swabbi"-object and not only to the original object.

confidence: Trust is used to measure the reliability of a specific peer. The confidence-attribute returns a figure describing a specific statement. Again it is stored as a number between 0 and 1.

additionDate: This attribute keeps track of the date it was added to the local repository. This could be used to determine confidence; old information might become less reliable. The main reason for this will be to make the right updates.

security: Security issues and access rights are important in enterprizes. In the wide open Peer-to-Peer environment some access control is required to ensure proper usage of the information. The security mechanism is not yet specified in detail, but is provisioned with this attribute.

visibility: For editing purposes some objects will have to be hidden, instead of being completely removed. This can be achieved by this attribute. Furthermore, the cached information is annotated with this property and set to "false".

cache: To increase the network efficiency caching of information will be necessary. The cached information is annotated with this property and set to date of the inclusion.

5 Working with the Model

In order to use the model described above for semantics-based information exchange, we have to provide a set of methods for constructing the repository according to the metadata model and to access the knowledge that is stored therein. In the following, we describe methods that have been developed to (1) create repository content mostly automatically from local information source, to (2) rank information in a repository based on the confidence we have in its reliability and to (3) access parts of the content in the repository and present it to the user in a comprehensive way.

5.1 Creation

Building up the Model The creation of the metadata comprises the extraction mechanism, the metadata integration and the ontology creation.

Extraction As mentioned above, the SWAP-system provides an integrated model of different structures that coexist in the network and the peer itself. These structures are file systems, emails, databases, ontologies and others. In order to include the different information sources into our model we extract an RDF(S) representation from them. This extraction is just on a syntactical level, i.e. the existing structure is translated into an RDF(S) representation. The single information items are annotated according to an information source specific metadata model. This metadata model has to be built for each information source manually. However, the metadata model does not model information which could not be extracted automatically from the information source.

Note, that the use of the SWAP-system is not restricted to SWAP-peers which have used the *swap-common* namespace. It represents the peer's knowledge about the network. As long as the other peer understands the query language it can participate as an information provider and seeker.

```
<rdf:RDF xmlns:rdf='http://www.w3.org/1999/02/22-rdf-syntax-ns#'
    xmlns:rdfs='http://www.w3.org/2000/01/rdf-schema#'
    xmlns:swapcommon="http://swap.semanticweb.org/2003/01/swap-common#"
    xmlns:swap="http://swap.semanticweb.org/2003/01/swap-peer#">

<swapcommon:Folder
        rdf:about="swap://1234567890.jxta#project">
    <rdfs:label xml:lang="en">Project</rdfs:label>
    <swapcommon:location>
        filefolder://windows/c:/Project
    </swapcommon:location>
</swapcommon:Folder>
</rdf:RDF>
```

Metadata integration Having built an RDF(S) representation of the information sources, in the second processing step the "Swabbi"-objects are added. Adding the metadata to the extracted structures is straightforward. A new object is created for each resource or statement. To link a "Swabbi"-object to a statement we use the construct of reification.

The properties (as presented in 4.1) are filled accordingly and a reference between the resource and the “Swabbi”-object is established with a “hasSwabbi” relation. Some of the “Swabbi”-object’s properties are now examined in more detail. The “additionDate” property is set to the date when the statement was added or updated. The “visibility” property is set to “true” if the user included the statement or resource and wants it to be displayed otherwise “false”.

One major source of information are other peers. Query results are returned as an RDF-graph. The user selects the information to keep. This information is added to the local repository in the same way as any other information from the peer itself. Of course the “hasPeer” property of the “Swabbi”-object changes. The “peerTrust” property might alter when receiving queries. This mechanism is described in the next section.

The example shows a reified statement with its corresponding “Swabbi” information [2].

```
<!--   peer1  -->
<rdf:Statement
        rdf:about="swap://1234567890.jxta#statement01">
    <rdf:subject rdf:resource="swap://1234567890.jxta#project"/>
    <rdf:predicate rdf:resource="rdfs:subclassOf"/>
    <rdf:object rdf:resource="swap://1234567890.jxta#thing"/>
    <swap:hasSwabbi rdf:resource="swap://1234567890.jxta#swabbiObjectNo01"/>
</rdf:Statement>

<swap:Swabbi
        rdf:about="swap://1234567890.jxta#swabbiObjectNo01">
    <swap:hasPeer  rdf:resource="swap://1234567890.jxta#knownPeers0001" />
    <swap:label>Project</swap:label>
    <swap:uri rdf:resource="swap://1234567890.jxta#project" />
    <swap:location>filefolder://windows/c:/Project</swap:location>
</swap:Swabbi>

<swap:Peer rdf:about="swap://1234567890.jxta#knownPeers0001">
    <swap:peerId>1234567890</swap:peerId>
    <swap:peerLabel>Christoph</swap:peerLabel>
</swap:Peer>
```

Ontology creation The extracted structures are a starting point for the ontology creation process. We suppose that the extracted information and background knowledge like e.g. WordNet[3] can be used to create these richer structures. This would mean adding formerly implicit semantics explicitly to the structure.

We assume for the moment, that we can determine the type of a certain information item in an ontological sense and can describe it as either concept, instance or property. If we extract for example a folder with the label “Project” we assert it is a concept while the label “SWAP”[4] would rather correspond to an

[2] Just differences are shown
[3] http://www.cogsci.princeton.edu/ wn/
[4] SWAP is a project of the EU. Contract no. EU IST-2001-34103.

instance of this concept. To represent this kind of statements in RDF we use the multiple inheritance mechanism of the language. In this manner we just have to add a new "rdf:type" statement to the already existing resource.

Automatically made assertions will probably lead to contradictions within the model. With the "confidence" we can adjust our believe in the particular statement.

In the following example just the changes in the structure are shown:

```
<!--   peer1  -->

<rdf:Description
         rdf:about="swap://1234567890.jxta#SWAP">
    <rdf:type rdf:resource='swap://1234567890.jxta#project'/>
</rdf:Description>
```

Merging One goal of the SWAP system is to have *one* knowledge representation. Through addition of resources and statements from other peers the same object might be present in the local repository, but under different names. The system has to identify these two objects through similarity measures and merge them accordingly. Whereas in RDF(S) no explicit relation as "equals" is defined, we use the equality relation defined in OWL[5].

Rating Model Content Statements (subclassOf and instanceOf) made by peers can be incomplete, vague or even false. For this reason, statements are not accepted by a peer as an absolute truth, but is judged based on the previous experience with the sender [6]. For example, if the sender tells the receiver something about gorillas and the receiver knows that the sender is an expert on orangutans, then it can derive from the fact that both concepts (gorilla and orangutan) have a small semantic distance that the sender probably also knows more than an average user about gorillas. To formalize the expertise we introduce confidence ratings that are meta-statements placed in the 'Swabbi' that indicate the confidence in a certain statement. The rating methodology involves the following aspects: (1) assigning confidence ratings to statements received by different peers. (2) calculating the confidence in the total statement itself (3) updating these ratings when new information from these peers is received. (4) using these ratings to determine a set of peers that have a high probability of correctly answering an incoming query and (5) applying an aging mechanism on ratings and removing statements that have a rating below a certain value. We describe these aspects in more detail:

Assigning confidence ratings to statements from a peer In this case we have to distinguish between derived statements from the extraction algorithm described in the previous paragraph and statements received from external peers. In the first case we assume that the user is confident in the statements that are derived and therefore assigned a high confidence rating. The exact value has to be determined by experiments. In the second case the confidence ratings are calculated from the previous statements from the sender. When a statement is provided by an unknown source it gets a (low) initial confidence rating. With 'unknown' we mean that the source never has provided any information to the receiver before. Thus, when a peer a receives information from peer b and b is unknown to a, then the statement from b gets a (low) initial confidence rating. If, however b already provided statements before to a then out of these statements the new confidence is calculated. The value will be a weighted average where the weighting factor is determined by semantic distance between an old statement and the new one. The similarity measure we will use is adapted from [7].

Updating confidence ratings If other peers than the original sender confirm the statement by repeating it, the statement gains higher confidence. The amount of gain depends of the confidence in the confirming source. This recursive definition of rating is also used in PageRank in Google [8] where the rank of a source depends on the ranks of the sources voting for that source. An important side effect of updating the confidence ratings is that also the confidence in the expertise of the original sender automatically increases.

Determining the experts to be queried When a query is received, the receiver first tries to answer the query itself. If it doesn't have a satisfying answer, it tries to find experts on the topic of the query. The system tries to find experts on topics that have a close semantic distance to the topic of the query. Again, we use the similarity measure described above for this.

Aging mechanism to devaluate confidence ratings in time A SWAP peer can retrieve large set of statements from other peers and from the generated statements by the ontology extractor. To keep the local repository scalable, we use an aging mechanism that removes statements that are too old in combination with a low rating. Experiments have to adjust the right parameters for tuning the way when and how to remove statements.

5.2 Access

A problem of the model as described so far is the fact that accessing it becomes quite inconvenient as the model does not only contain the actual knowledge but also management metadata. Our solution to this problem is to provide the user of the SWAP system with a set of definable views on the repository. The views are defined using the management metadata, but they do not contain it any more. Being stripped of the metadata a view can be treated like an ordinary RDF model of a specific part of the Peers knowledge.

In the following, we describe the SeRQL query language we use to define views over the knowledge repository of a peer. We discuss features for selecting parts of the repository and for constructing new RDF models from these parts. Furthermore, we describe mechanisms for persistently defining views as a basis for providing the user with a visualization of a peers knowledge from a particular point of view.

The SeRQL Query Language SeRQL is an RDF query language that has been developed on the basis of experiences with implementing and using different state-of-the-art query languages such as RQL [9] and RDQL[5]. The language is currently being implemented in the Sesame System [10] which is used to store the knowledge of peers in SWAP. The main feature of SeRQL that go beyond the abilities of existing languages is the ability to define structured output in terms of an RDF graph that does not necessarily coincide with the model that has been queried. In contrast to QEL [11] it has a comprehensive syntax. This feature is essential for defining personalized views in the repository of a SWAP peer. Before coming back to this issue, we will first discuss the selection mechanism used in SeRQL and its application in the context of selecting parts of the knowledge in the repository based on attached metadata.

The selection mechanism of SeRQL is based on path expressions borrowing from existing languages. RDF triples are seen as arcs in a graph where combinations of triples form paths in that graph. Path expressions are now expressions that describe certain types of paths, a trivial graph only consisting of a single triple will be represented as {`resource`$_1$} `property` {`resource`$_2$}. This path expression can already be used to extract content from the repository. For example, we could ask for all classes in a model using the expression {`X`} `<rdf:type>` {`<rdfs:Class>`} where `X` is a variable. In the course of querying this variable will be instantiated with all resources in the queried model that are of type `<rfds:Class>` which are returned as a result.

In order to formulate more complex selection criteria, the simple path expressions can be combined in different ways to describe subgraphs in the RDF model. Typical examples (compare model in figure 2) are:

reification: All subclass-relations and assigned swabbies:
  `{{X} <rdfs:SubClassOf> {Y}} <swap:hasSwabbie> {S}`
sequence: Things that have a Swabbi with the visibility set to true:
  `{X} <swap:hasSwabbi> {} <swap:visibility> {"true"}`
split: "visible" Swabbies with location C:/Projects/SWAP:
  `{X} <swap:visibility> {"true"}; <swap:location>`
  `{"C:/Projects/SWAP"}`
join: Swabbies that refer to the same location:
  `{X,Y} <swap:location> {Z}`

The last example will also return the result where the two variables `X` and `Y` are bound to the same resource. In order to prevent situation like this and to more general to define equality and inequality of variables, SeRQL allows to specify Boolean restrictions on variable binding. In the example we could add the restriction `X != Y` to state that we are only interested in pairs of non-equal Swabbies sharing the same location. Furthermore, we could replace all directly mentioned resources in the expressions above by variables and add restrictions

[5] http://www.hpl.hp.com/semweb/rdql.html

to the query that enforce these variables to be equal with a certain resource or literal.

View Definitions One of the main new features of SeRQL is the ability to not only retrieve tuples of resources that match a selection statement, but to return a complete RDF model as a result of a query. For this purpose, a SeRQL query can be accompanied by a construction part that specifies an RDF graph. The specification consists of a path expression that shares some variables with the selection part of a query. Whenever the selection part matches a subgraph in the queried model a new RDF graph is created by instantiating the creation part with the values that have been bound to the shared variables.

Using SeRQLs creation mechanism, we can extract certain views from the repository of a peer. The most basic construction mechanisms is to create a copy of the matched parts of the repository. This mechanism can be used to extract the real knowledge from the repository without the attached metadata. The following expression for example extracts the subclass hierarchy and the instances from a SWAP repository:

```
CONSTRUCT *
FROM
    {C} <rdf:type> {<rfds:class>}
    {I} <rdf:type> {C}
    {X} <rdfs:subClassOf> {Y}
```

One of the reasons of attaching metadata to the knowledge in the repository, however is to provide criteria for extracting certain parts of the knowledge in a repository. This could be the part of the knowledge that is actually trusted or the knowledge that is provided by a certain peer. This kind of selective extraction of knowledge needs more sophisticated construction statements. The following definition selects the subclass hierarchy that is provided by the external peer with the label 'peer42':

```
CONSTRUCT
    {X} <rdfs:subClassOf> {Y}
FROM
    {{X} <rdfs:subClassOf> {Y}} <swap:hasSwabbie> {}
    <swap:hasPeer> {} <swap:hasLabel> {"peer42"}
```

Beyond this, we also have the possibility to define a view with no direct structural correspondence with the content of the repository by inventing new properties in the construction part. Using this possibility we can for example create a view that describes the expertise of known peers. The following definition for example creates a model that describes the concepts certain peers know about:

```
    CONSTRUCT
        {P} <view:knowsAbout> {C}
    FROM
        {{C} <rdfs:type> {<rdfs:Class>}} <swap:hasSwabbie> {}
        <swap:hasPeer> {} <swap:hasLabel> {P}
```

Summarizing, we can say that the construction abilities of SeRQL provides us with a powerful mechanism to extract information to be presented to the user from the content of a SWAP repository. The metadata model chosen in the SWAP project provides the necessary background information for the definition of meaningful views that can assist the user in finding and assessing information.

Storing Views The first thing, we observe when looking at the descriptions so far is the inability to explicitly refer to the result of query. This in fact is a necessary requirement for the use of a query as a view as the result of a view definition should be handled like any other RDF model. RDF models are referred to using a namespace URI. Our first extension to SeRQL is therefore a construct that makes it possible to assign a target URI to a query that represents the virtual model created. The second important point about a view definition is that their result should be another RDF structure. Therefore, view definitions will always contain a construct part instead of a select part. Furthermore, we need to be able to explicitly represent a query in an RDF repository as part of a model. For this purpose, we propose the following simple RDF serialization of views in SeRQL:

```
<serql:view ID="ExampleView"
    target="http://sesame.aidministrator.nl/ExampleView/">
    <serql:description>
        Just an example of a view definition without content.
    </serql:description>
    <serql:expression>
        <serql:useNamespace>...</serql:useNamespace>
        <serql:construct>...</serql:construct>
        <serql:from>...</serql:from>
        <serql:where>...</serql:where>
    </serql:expression>
</serql:view>
```

This simple model suffices for our purposes as it assigns an ID to a query in the same way it is done for any other RDF resource and a target name space that will be used to refer to the virtual RDF model that is defined by the view. Furthermore, we include a description part that is supposed to contain a free text description of the provided view which is mainly meant for maintenance purposes. The second main part of a view definition is a query expression in the SeRQL language. The only modification we make is to introduce RDF properties for the different parts of a SeRQL query. These properties are introduced because they make it possible to enforce that a view definition actually contains a construct part that can be referred to when accessing data in the view. The values of these properties are just valid SeRQL expressions for the different parts as defined in the SeRQL grammar. Reference to such a view definition can now be made via its target URI. For example:

```
http://sesame.aidministrator.nl/ExampleView
```

This simple introduction of an RDF container for a SeRQL query expression already provides us with a view definition mechanism.

Visualizing Views To achieve the main goal of the SWAP system, i.e. satisfying the information needs of the users, access to information must be easy and understandable. As seen before, the view building mechanism already offers a flexible way to select parts of the available information and to render them in constructs pre-defined by the user. However, these views are RDF files which are very difficult to understand for human users. Visualization plays an important role in making these views understandable. The employed visualization techniques depend on the complexity of the views. We distinguish two major scenarios.

Simple Views We expect many of the views to be instantiated light-weight ontologies, i.e. concept hierarchies and their instances. Note that this information (classes, their hierarchy and instances) can be obtained with SeRQL queries such as our first query example. We think many views will be light-weight because of two facts. First, many information systems rely on light-weight ontologies. Also known as taxonomies, such ontologies are frequently used in several domains (biology, chemistry, libraries) as classification systems. Information architects consider taxonomies as basic building blocks, representing the backbone of most web sites. Non-formal taxonomies are already widely used in web applications for product classification (e.g. Amazon) or web-directories (e.g. Yahoo, Open Directory Project (ODP)). Second, the information sources of the peers are often semantically shallow: folder structure, email structures, databases. Current ontology extraction methods are only capable of producing light-weight ontologies from such sources.

The Cluster Map [12] technique was specially developed for visualizing instantiated concept hierarchies. Therefore we will use it for visualizing the simple views. A Cluster Map depicts all the concepts, their hierarchy as well as their instances. Fig. 3 shows a collection of job offers organized according to a very simple ontology. Each small yellow sphere represents an offer (an instance). The big green spheres represent ontology concepts, with an attached label stating their name and cardinality. Directed edges connect classes, and point from specific to generic (e.g. *IT* is a subclass of *Job Vacancies*). Balloon-shaped edges connect objects to their most specific class(es). Objects with the same class membership are grouped in clusters.

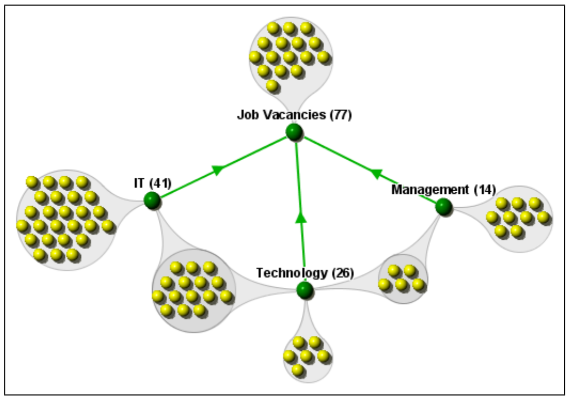


Fig. 3: The Cluster Map of a simple view.

This visualization shows two common characteristics of instantiated taxonomies which are difficult to represent textually: *incomplete classes* and *overlaps*. The set of subclasses of a class is *incomplete* when their union does not contain all the objects of the superclass. Indeed, in our example, the subclasses of the root class are incomplete as their union does not cover their superclass: some members of *Vacancies* were not further classified. Classes that share instances are *overlapping* if no specialization relationship holds between them. For example *Technology* and *Management* overlap.

Cluster Maps support several user tasks such as analysis, query and navigation of semantic structures [12]. Of all these, analysis is the most beneficial for the SWAP system. There are many ways to do analysis. First, one can inspect the same data source from different perspectives by visualizing it according to different views. For example, a set of web-pages describing job vacancies can be presented through at least two views. A first view can contain all the sub-concepts of the "Geographic-location" concept and result in the visualization of the vacancies according to their geographic distribution. Yet another view can be constructed from concepts describing the economical sector relevant for the job offer (see Fig. 3). Second, different data sets can be visualized according to the same view, allowing for a comparative analysis the data sets. Figure 4 compares the activity profiles of two banks by visualizing the pages on their web sites according to the same view. To get such data as in Fig. 4 it is enough to instantiate a construct structure that extract concepts and their instances (a simplified version of the first example query).

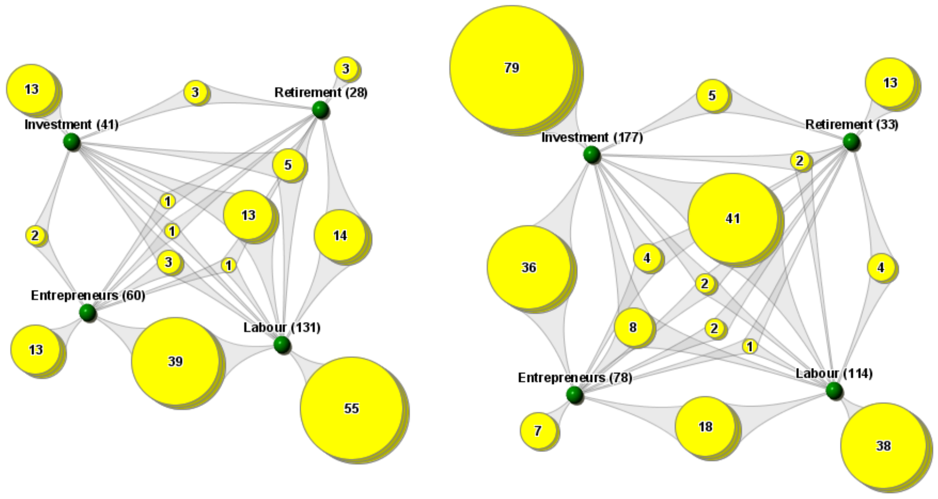
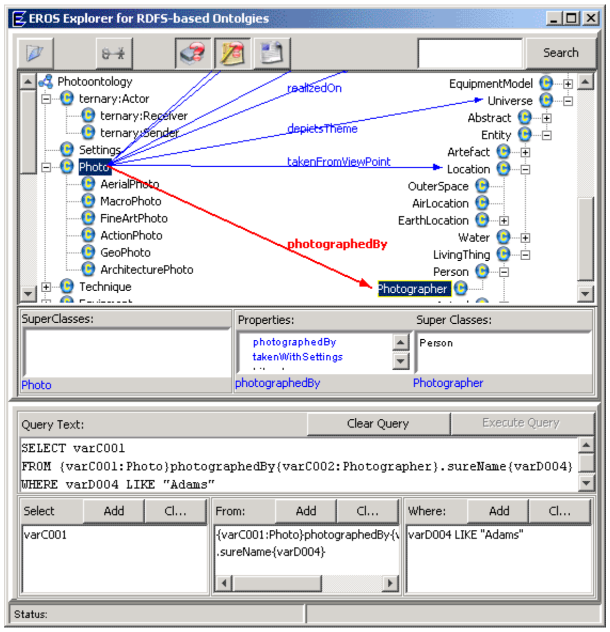


Fig. 4. The web-pages of two banks according to the same view.

Complex Views. By using the CONSTRUCT part of the SeRQL queries it is possible to extract information in complex views defined by the user. This was demonstrated in the third example SeRQL query which introduces a new property (knowsAbout). The returned answer for this query will contain a set of resources linked by properties, a structure which cannot be displayed with the Cluster Map. Of course, this is just a simple example: SeRQL allows defining much more complex RDF structures.

We chose two types of visualizations for depicting views with different complexity: a tree-based technique is employed for simple concept hierarchies and a graph-based technique is used to visualize complex RDF structures. As discussed before they both have their strength and weaknesses. As future work we consider adapting techniques that combine the benefits of both approaches. For example the EROS[13] system which was specially designed to depict complex structures in such a way that (1) the concept hierarchy is visible and, meanwhile, (2) properties can be depicted as well.

We chose two types of visualizations for depicting views with different complexity: a tree-based technique is employed for simple concept hierarchies and a graph-based technique is used to visualize complex RDF structures. As discussed before they both have their strength and weaknesses. As future work we consider adapting techniques that combine the benefits of both approaches. For example the EROS[13] system which was specially designed to depict complex structures in such a way that (1) the concept hierarchy is visible and, meanwhile, (2) properties can be depicted as well.



SuperClasses: Photo

Properties: photographedBy, takenWithSettings — photographedBy

Super Classes: Person — Photographer

```
SELECT varC001
FROM {varC001:Photo}photographedBy{varC002:Photographer}.sureName{varD004}
WHERE varD004 LIKE "Adams"
```

Select: varC001

From: {varC001:Photo}photographedBy{v .sureName{varD004}

Where: varD004 LIKE "Adams"

Fig. 5. The EROS interface.

6 Conclusion

The completely distributed nature and the high degree of autonomy of individual peers in a P2P system comes with new challenges for the use of semantic descriptions. If we want to benefit from the advantages that normally accompany the use of ontologies as specifications of a shared vocabulary we have to find ways to dynamically align the semantic models of different peers. In this paper, we described a model that combines features of traditional ontologies with rich metadata about the origin of information and the reliability of sources. Furthermore, we introduced methods for creating, assessing and accessing semantics and metadata.

Our model has several advantages over traditional ontologies in the context of Peer-to-Peer information exchange. The most important feature with this respect is the fact that statements in the semantic models are not seen as being the truth as in most traditional models. We rather see the semantic model as a collection of opinions supported by different sources of information. Opinions many sources agree about are more likely to be true than opinions that are not shared across the system or that even contradict other opinions. This makes it possible to directly extract semantic models from information sources even if these are not completely compliant with the existing model. Furthermore, we can use heuristic methods to align and update semantic models. If the result of such an update is shared by many peers it will persist, if not it does not harm the system.

A key issue for the acceptance of such heuristic methods of course is a careful evaluation of their performance in general and in concrete applications. In order to evaluate the model and the methods on a general level, test procedures are developed in the SWAP project that use simulation techniques to experiment with large scale Peer-to-Peer systems [14]. Furthermore, case studies in using the SWAP system in the tourism and the finance domain are planned. These case studies will show whether the general ideas really provide benefits in real world applications.

One of the most fundamental questions that has to be answered by the case studies is whether it is sufficient to rely on structures that have been extracted from information sources instead of hand-crafted knowledge structures and metadata. It may turn out that in addition to the extraction approach, we also need to annotate information by hand. In this case we have to investigate how methods for supporting semantic annotation (e.g. [15]) can be integrated in the system in order to build up the knowledge structures described in this paper.

References

1. OLeary, D.: Using ai in knowledge management: Knowledge bases and ontologies. IEEE Intelligent Systems **13** (1998) 34–39
2. Gruber, T.R.: Towards Principles for the Design of Ontologies Used for Knowledge Sharing. In Guarino, N., Poli, R., eds.: Formal Ontology in Conceptual Analysis and Knowledge Representation, Deventer, The Netherlands, Kluwer Academic Publishers (1993)
3. Ehrig, M., Tempich, C., Broekstra, J., van Harmelen, F., Sabou, M., Siebes, R., Staab, S., Stuckenschmidt, H.: SWAP - ontology-based knowledge management with peer-to-peer technology. In Sure, Y., Schnurr, H.P., eds.: Proceedings of the 1st National "Workshop Ontologie-basiertes Wissensmanagement (WOW2003)". (2003) To appear 2003.
4. Broekstra, J., Klein, M., Decker, S., Fensel, D., van Harmelen, F., Horrocks, I.: Enabling knowledge representation on the web by extending rdf schema. In: Proceedings of the tenth World Wide Web conference WWWW'10, Hong Kong (2001)
5. Dean, M., Connolly, D., van Harmelen, F., Hendler, J., Horrocks, I., McGuinness, D.L., Patel-Schneider, P.F., Stein, L.A.: Owl web ontology language 1.0 reference, (Internet:http://www.w3.org/TR/owl-ref/)
6. Siebes, R., van Harmelen, F.: Ranking agent statements for building evolving ontologies. Proceedings of the AAAI-02 workshop on meaning negotiation, Alberta, Canada (2002)
7. Resnik, P.: Semantic similarity in a taxonomy: An information-based measure and its application to problems of ambiguity in natural language. Journal of Artificial Intelligence Research **11** (1999) 95–130
8. Page, L., Brin, S., Motwani, R., Winograd, T.: The pagerank citation ranking: Bringing order to the web. Technical report, Stanford Digital Library Technologies Project (1998)

9. Karvounarakis, G., Alexaki, S., Christophides, V., Plexousakis, D., Scholl, M.: Rql: A declarative query language for rdf. In: The 11th International World Wide Web Conference. (2002)
10. Broekstra, J., Kampman, A., van Harmelen, F.: Sesame: An architecture for storing and querying rdf data and schema information (2001)
11. Nejdl, W., Wolf, B., Qu, C., Decker, S., Sintek, M., Naeve, A., Nilsson, M., Palmér, M., Risch, T.: Edutella: A P2P networking infrastructure based on rdf. In: Proceedings to the Eleventh International World Wide Web Conference, Honolulu, Hawaii, USA (2002)
12. Fluit, C., Sabou, M., van Harmelen, F.: Ontology-based Information Visualisation. In Geroimenko, V., ed.: Visualising the Semantic Web. Springer Verlag (2002)
13. Vdovjak, R., Barna, P., HOuben, G.: EROS:Explorer for RDFS-based Ontologies. In: Proceedings of Intelligent User Interfaces, Miami, Florida, USA (2003)
14. Ehrig, M., Schmitz, C., Staab, S., Tane, J., Tempich, C.: Towards evaluation of peer-to-peer-based distributed knowledge management systems. In van Elst, L., Dignum, V., Abecker, A., eds.: Proceedings of the AAAI Spring Symposium "Agent-Mediated Knowledge Management (AMKM-2003)". Springer LNAI, Stanford, California, Stanford University (2003) To appear 2003.
15. Handschuh, S., Staab, S.: Authoring and annotation of web pages in cream. In: Proceedings of the 11th International World Wide Web Conference, WWW 2002, Honolulu, Hawaii, May 7-11, 2002, ACM Press (2002) 462–473