Gnowsis Adapter Framework: Treating Structured Data Sources as Virtual RDF Graphs

Leo Sauermann, Sven Schwarz

Knowledge Management Department
German Research Center for Artificial Intelligence DFKI GmbH,
Kaiserslautern, Germany
and
Knowledge-Based Systems Group, Department of Computer Science,
University of Kaiserslautern, Germany
{leo.sauermann, sven.schwarz}@dfki.de

Abstract. The integration of heterogenous data sources is a crucial step for the upcoming semantic web – if existing information is not integrated, where will the data come from that the semantic web builds on? In this paper we present the gnowsis adapter framework, an implementation of an RDF graph system that can be used to integrate structured data sources, together with a set of already implemented adapters that can be used in own applications or extended for new situations. We will give an overview of the architecture and implementation details together with a description of the common problems in this field and our solutions, leading to an outlook on the future developments we expect. Using our presented results, researchers can generate test data for experiments and practitioners can access their desktop data sources as RDF graph.[1]

1 Introduction

Semantic Web applications have a need for data to work on, and to access existing data sources like file systems, relational databases or legacy applications to extract information and represent it in RDF. The gnowsis adapter framework offers an architecture for adapters that do exactly this task. The framework consists of three parts, first an abstract implementation of adapters that build a skeleton for concrete adapters, second a middleware to integrate multiple adapters into a single, queryable data source and third a set of reference implementations that show how to build adapters that can be used off the shelf. Using this framework, application developers can extract information from heterogenous data sources based on the existing off-the-shelf-adapters or own implementations. The main features of the framework are:

– Interfaces and abstract implementations of adapters

[1] This work was supported by the German Federal Ministry of Education, Science, Research and Technology (bmb+f), (Grant 01 IW C01, Project EPOS: Evolving Personal to Organizational Memories).

- Existing adapter implementations to access the local filesystem, IMAP email servers, Microsoft Outlook, relational databases, and the Mozilla address book as RDF graphs
- A framework to configure, start and integrate several adapters
- Instructions and Examples to build custom adapters

The gnowsis adapter framework is based on the Jena RDF framework[11] and extends the existing graph API. Each adapter implements either the graph API or a method to create concise bounded descriptions (CBDs), a data exchange format defined by Patrick Stickler in the URIQA web service [15]. In the latter case these CBDs will implement the graph API indirectly. This means arbitrary graph handling can take place nevertheless. Other more specialized third-party adapters have also been integrated into the framework.

The functionality of the framework is restricted to extract information, changes on the underlying data are not supported. There is no clear specification how to express changes and send them to the adapter, especially facing blank nodes, as our focus lies on generating data for the semantic web we decided to start with an extraction framework and continue our work when the SPARQL [7] specification contains a methodology for updates and after reference implementations exist.

In this paper we will describe the implementation and use of the system, together with references to related projects. We will show the necessary steps to create, deploy and use the adapters. The main benefits of using such adapters together with the underlying framework will also be explained. An evaluation has been made regarding (a) the *implementation cost* of adapters (by measuring the time a person takes to write an adapter) and (b) the *performance* of the different adapters, using different query methods and different data sources.

2 Structured Data Sources

For our work we focus on RDF extraction from structured data sources, with a distinction between *files, applications and RDF repositories*. We need these later, when deciding what kind of adapter to implement with what functionality.

2.1 File Data Sources

There exist many popular, standardized file formats, that contain valuable structured data for the semantic web. Usually there also exist tools, which extract the structured information and convert it into RDF conforming to some popular RDF schema(s). Examples of some interesting file formats are:

- *iCalendar* [6] is a popular text format for calendars and consequently used by the MacOs calendar and popular open source calendaring tools. It is an exchange format between calendar applications, and converters for RDF exist as a result of the RDF Calendar Workspace[2].

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

- *JPEG Images* are the default file format used by digital cameras and are also popular on the WWW. The format allows embedded meta-data in the EXIF format [3]. There exist several tools [1] to extract EXIF from image files and already represent the result as RDF according to a RDF-Schema [10].
- *PDF documents* contain structured information about author, creation date and much more information can be embedded using the RDF-based XMP format.
- *HTML files* contain meta-information in the head of the document, structured meta-data about author, keywords, etc.

We concentrate on files that have characteristics of documents, not large storage files like relational database files or excel sheets.

2.2 Application Data Sources

These are systems that hold structured data in a known format and are accessible from outside to extract the data. Some are in popular use and can be easily integrated into the RDF.

- *Relational Databases* that implement the SQL standard are a typical example. For these, bindings in all common programming languages exist and different implementations offer the same behavior. Many web applications use MySQL to store their information, offering a vast pool of information that can be transformed. D2RQ [5] by Chris Bizer, Andy Seaborne and Richard Cyganiak is a generic tool that represents relational data as RDF graph, by transforming RDF queries into SQL statements and then converting the query result to RDF. We used this tool as an example of how to write adapters, our results are written below in the evaluation section.
- *IMAP email servers* are a common way to store emails and access them using different email clients. The protocol is defined as RFC and many existing APIs allow access to the servers. Emails are an important factor in today's knowledge work.
- *Microsoft Outlook* and the corresponding Exchange Server are a common collaboration tool in business environments. The interfaces to the Outlook client are well defined and many applications integrate with Outlook. An RDF adapter for Outlook was already described in [13] and will serve as an example.

For these and other application data sources, generic adapters would be useful to access them through RDF. The cost of implementing an adapter is justified by the possible reuse in many scenarios.

2.3 RDF repositories and web-services

Finally, existing RDF data sources published through proprietary or web interfaces are very interesting, as they already implement the RDF framework.

Typical representations are Sesame [4], or kowari[3]. Integration of these has already been discussed in [9] but has to be reviewed again in the current situation of the upcoming SPARQL protocol and the integration with other data sources.

3 Gnowsis Adapter types

Definition 1. *We define* adapters *as a software tool that can, on request, extract data from existing structured data sources and represent them as RDF.*

The principle of adapters is well known in software architecture, as a design pattern described in [8] and as a data extraction component in many applications areas, ie search engines. To access data sources, we can differ between three basic approaches, each suiting a certain purpose and having advantages and drawbacks. We identified the following approaches to build an adapter:

3.1 Graph and Query Adapters

An adapter that implements the interface of an RDF graph or a sophisticated query language like RDQL, SPARQL or TRIPLE. Typically, these interfaces are based on querying the graph with a method that can be described with a *find (Subject Predicate Object)* interface. The adapter has to answer to queries, each query consists of a pattern matching the graph. The output of such an adapter would be a list of statements that match the query. Both the SAIL API of Sesame and the Jena Graph API offer a comparable architecture to a *graph adapter.*

3.2 Concise Bounded Description (CBD) Adapters

This adapter can return a small subgraph that describes exactly one resource in detail. The exact definition of the result is given as part of the URIQA specification. There, a *Concise Bounded Description* (CBD)[15] is the basis for communication to web services. The interface such an adapter implements would be: *get the CBD of resource with URI X.* The source on which it works could be any application data source or the files stored on a web server. CBD adapters do not depend on a complicated query framework neither can they answer any other request, but they serve as bootstrapping interfaces to quickly retrieve information about a single resource.

3.3 File Extractors

Finally, extraction of meta-data stored in files requires special handling. Typically, a file extractor is a tool that reads files, parses them and returns some meta-data that was extracted from the data stream. An example would be an

[3] http://www.kowari.org/

EXIF extractor that reads JPEG files and returns an RDF description of the metadata. A typical use case for a file extractors would be a search engine that builds and index over the metadata of files, another example is a Semantic Web browser that can show the content of a file together with the metadata. These usage scenarios require that file extractors have to be lightweight, easy to program and have a fast execution time.

For above approaches we have to discuss the benefits and drawbacks, you will find these below in the evaluation section.

4 How to Build Gnowsis Adapters

Writing an adapter requires a series of important tasks. Aiming at a complete methodology we successively collected information from our own experience and by observing the RDF-Calendar working group of the W3C. The overall task of building an adapter works according to the following main top-level tasks:

1. Analyze the data source
2. Choose an appropriate RDF representation
3. Choose an appropriate adapter type
4. Implement the adapter

Finally, different adapters can be integrated in a *middleware* architecture [9] for RDF repositories, so some additional configuration and deployment tasks have to be done. We implemented a specialized architecture for the integration of desktop data sources. All the above mentioned tasks will now be explained in detail.

4.1 Analyzing the Data Source

First, as a preparation, the data source in question has to be analyzed. You have to find some test data for development of an adapter. A documentation of the file format is needed or a documentation of the API how to access the data source, if it is an IMAP server or a database. Normally, existing tools can be examined that already implement the extraction mechanism and, sometimes, a tool exists that already returns the structured data as RDF.

- Formal Description of the data source. For IMAP this would be the RFC, for Microsoft Outlook it is the Programmer Reference, for iCal it is the RFC.
- Example Source Data. Valid data that has been generated by common applications.
- Search for an existing implementation to extract the information from the data source, preferably one that returns RDF.

4.2 Formal RDF Representation

If two adapters extract the same information but return different RDF representations, or use different identification schemes for resources, the usability suffers.

For most data sources, no formal representation of the data in RDF exists. In 2004 we did a survey on `www.schemaweb.info` and found only one public data format for representing Emails using RDF-Schema, and it was not in popular use. So, first is the search for an existing RDF-Schema that can represent the data. When an RDF-Schema formalization of the data exists, it should be used. If two exist, take the more popular scheme, we recommend the comparison of search results on google (`www.googlefight.com`), using the namespace identifier of the vocabulary as a search term. From our experience, it is always better to extend existing schemas to represent additional statements, than to build a new vocabulary from scratch. The FOAF project is a good example how a simple vocabulary can be extended for special needs.

We discourage the use of OWL-FULL as a formal representation of data, as there are issues regarding the reasoning implementation of this standard. RDF-Schema in combination with OWL-DL will, in most cases, provide enough means to represent the needed classes and properties of a structured data source.

Then, a solution for the *identification of resources* has to be found. Ideally, the data source already uses URIs to identify resources or a specification exists that explains how to identify resources. For our IMAP adapter the RFC [12] gives such a specification. For relational databases, D2RQ defines a way how to use primary keys to create meaningful URIs. Additionally, you have to regard scheme specific requirements, e.g., for the popular FOAF vocabulary, it is best practice to identify human persons by using the email address as inverse functional property. The resulting data should be compatible to already existing RDF representations, so common practice is as valuable as the formal definitions.

When the formal representation in RDF-Schema or OWL is found and the approach to identification is known, *test data* in RDF representation has to be authored. Again, using examples that are in public use will help standardization.

4.3 Adapter Architecture Decision

Based on the nature of the data source, the intended usage scenario and other facts like programming experience or the existing software architecture, an architecture for the implementation can be chosen.

In the web we find a vast scenery of existing adapters, beginning at command line tools written in scripting languages that take input on stdin and return output on stdout to servers that specify their own protocols. We hope that above classification into three adapter types helps picking a suitable implementation path and making adapters reusable.

In table 1 we give an overview of architectures and facts that influence the decision.

	graph adapter	CBD adapter	file extractor
usage	RDQL, find(spo)	getCBD(uri)	getGraph(file)
data sources	application	application, file	metadata from files
implementation	based on existing API (Jena, SPARQL)	independent	independent
implementation cost	high	low	low
full RDQL support	yes	no	no
CBD support	yes	yes	yes

Table 1. A comparison of the different adapter types.

4.4 Implementation of an Adapter

Each adapter type requires a different approach to the implementation. While graph adapters can benefit by integration into an existing RDF framework, a file extractor can be implemented only returning an RDF text serialization. During implementation we recommend to continually test against the data sources and an example RDF file. *Testing against sample data* is especially helpful when working in a distributed team or when the implementation is delegated to untrained personnel.

Graph Adapters The gnowsis framework offers an abstract implementation of graph adapters. The concrete adapter class is then a subclass implementing the skeleton. Three abstract classes have to be implemented, the adapter itself, a set of resource wrappers and a set of property wrappers. For each resource type in the data source, a wrapping class is needed that represents the resource (and should buffer the source object). For each property, a property wrapper has to be implemented, with the functionality to generate triples. At this point we need the RDF-Schema or OWL formalization of the source's data format. This ontology is mapped to the wrapping Java classes.

The exact functionality of the graph adapters is described in [13, 2], together with more instructions on the implementation details, the existing adapters serve can serve as examples. Our approach to wrapping data sources using a mapping file from an ontology to java implementations is similar to the approach by Bizer et al [5].

CBD Adapters The main function of a CBD adapter is to return a subgraph around a named resource; a call to such an adapter will usually contain the URI of the resource and some parameters. The exact parameters are defined in [16], the gnowsis system only supports a subset of them, more information can be found in the documentation. The results returned by a CBD adapter in gnowsis are Jena Models, generated and filled by the adapter. In the result is the resource of the request together with the statements describing the resource.

The implementation is simpler compared to a graph adapter, it consists of parsing the URL, deciding how to generate the concise bounded description,

generating it and returning it. It does not involve dependencies to other parts of gnowsis nor does the implementation require a deeper understanding of the framework. An example for a simple CBD adapter can be found in the IMAP adapter implemented by Shen Jianping (it is part of the gnowsis distribution).

When adapting the data stored inside a third party application data source like the Mozilla Email client Thunderbird, there is another possibility to implement a CBD adapter. The adapter can be embedded into the application itself, generating the RDF representation in the serialized form (RDF/XML or N3) and returning it through an inter-process-communication interface like XML/RPC or activeX. The rather simple CBD interface and the URIQA protocol are easier to implement compared to a full RDF query language. We created such an adapter for Mozilla Thunderbird, it allows the access to data stored inside Thunderbird's address book through a CBD interface. The adapter was implemented mainly in Javascript as a Thunderbird plugin and is contacted by gnowsis.

File Extractors Similiar to CBD adapters, a file extractor returns the metadata of a single file as RDF. At the moment, we use an adapted CBD interface for file extractors: the call contains the file in question and the URI with which the file is identified. The extractor then uses the passed URI to generate the RDF representation of the file's meta-data.

We implemented an adapter that extracts the ID3 tags from MP3 files, they contain information about title, album, artist, ... Implementation of such adapters is straightforward and we do not need to describe the details here. The interesting question is, how to reuse existing file extractors implementations. For example, the iCalendar converters `fromIcal.py` and `ical2rdf.pl` created by Dan Connolly, Libby Miller, and Tim Berners-Lee provide simple and effective means to convert text files to RDF. They can be called from other applications as they take their input from stdin and return the RDF serialization on stdout. Many such converters and extractors exist for different file types, and as they implement similar interfaces, can be integrated to frameworks such as gnowsis. We hope that the interfaces defined by gnowsis are an aid for others.

4.5 Usage of Adapters

Typical usage of an adapter are requests for concise bounded descriptions (CBDs) or find(subject, predicate, object) calls. More complex query formats can also be implemented in adapters, but this would raise implementation cost and complex queries can be fractionated into several find calls, as it is done in the actual Jena implementation.

Jena (and other popular RDF APIs) require a find(spo) implementation. In complex queries (RDQL, SPARQL) including graph matching, the approach to query execution by the adapter could ...

- parse the whole query in its native form (RDQL) and execute it. The adapter therefore has to implement a query parser and an execution engine.

- only implement the find(spo) interface. The query parsing is done by another framework (for example Jena) and fractionated into several find(spo) calls. The result of the calls are aggregated by Jena. This works satisfyingly for most cases, if the client knows the structure of the graph and asks only find(spo) questions that will return an answer. In any case, a query reordering helps the execution engine. We implemented a basic query reordering algorithm in Gnowsis, extending the standard RDQL query engine of Jena.
- buffer all graph information in an RDF Repository. The framework has to crawl all adapters and store the crawled graph into one big database. We recommend using a Quad-Store or named graph store for this and chose the context id (the fourth resource in the quad or the name of the graph) the same as the CBD URI. Then, updates are on CBD identifier level.

4.6 Integration of Heterogenous Data Sources

Faced with several data sources, they have to be integrated to a hub architecture. [9]. In the gnowsis architecture we facilitated a registration of adapters and then a use of these adapters based on the registration information. Configuration of an adapter includes basic information about the implementation class (a Java class implementing an adapter type), human readable data source identifier, an ontology of the adapter and URI patterns that describe what URIs can be found inside the graph of the adapter. This configuration is, of course, stored in an RDF graph.

For crawling, each adapter has to define *root URIs*, one ore more URIs of a resource that is queried first from the adapter, the resulting resources of such querying (either by find(spo) or by CBD) can then be crawled further. The concept of root URIs can also be employed in user interfaces: the information content of the adapter can be explored starting at the root URI.

5 Benefits of the framework

Reusable adapters help to avoid reinventing the wheel every time some RDF data is needed. The obvious benefit of using standardized adapters is the reduced implementation effort in Semantic Web projects. The time to create and test custom implementations is higher compared to the time needed to integrate an existing adapter. If the existing adapter does not comply to performance requirements or functionality, it can be extended or used as a prototype.

The second, from our perspective far more important benefit, is the standardization process that happens when the data is transformed to RDF. If two distinct applications use different implementations to access the same data source, the two resulting RDF graphs may be different. We observed two main problems in data transformation, the used ontology and the resulting URIs.

The *ontology problem* can be illustrated by looking at the history of the W3C RDF-Calendar working group. A vocabulary to represent iCalendar event information in RDF was authored by Dan Connoly and others[4]. A sample script

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

(comparable to an adapter) was implemented by Tim Berners-Lee that showed the conversion. Then, the vocabulary was changed and the different implementations either implemented the old or the new version, resulting in confusion when a system used two adapters that converted the information differently. Another case would be, when the same data is transformed into two different ontologies, e.g., information about people can be expressed either in FOAF or in vCARD[5].

The *URI problem* is similar. By what URIs should resources be identified? We request that the same resource should be identified by the same URI. However, even for a simple resource like a local file this is not straightforward: The file URI is often implementation dependent. For instance, Mozilla and Internet Explorer use different URIs for the same local files when browsing.

Both problems can be avoided by always using the same adapter. Although automatic matching of ontologies is possible, it could be avoided when using the same extraction algorithm and ontology in the first place.

5.1 Deployment and use of adapters

The resulting adapters can be used in concrete applications or embedded in the gnowsis Semantic Desktop framework [14]. In the embedded scenario, the adapter will be packaged as gnowsis service in form of a WAR file (the standard servlet container format, extended by gnowsis). The WAR file is included into an installed gnowsis system and then configured using the options menu of gnowsis, resulting in the integration of the data source into the data sources that will be queried by gnowsis on browsing or searching features. The exact way of deploying an adapter and using it is described in the according tutorials on the project website [2].

Adapters can also be used independent of gnowsis, in any java application. For this, the gnowsis adapter framework has to be included as a dependency together with the complete Jena dependencies. To start and use an adapter, an RDF graph containing the configuration information for the adapter has to be passed during instantiation. Again, details on this can be found on the website. Altogether, using and deploying adapters inside and outside gnowsis is comparable (on the effort level) to other Semantic Web frameworks.

6 Evaluation

6.1 Implementation Cost

The task of implementing an IMAP adapter was delegated to one student. This student was an average skilled developer, who was new to gnowsis and RDF. He had to read and test the standard IMAP protocol, as well as, to read about and get into the gnowsis framework. During his work the student had to write down the amount of time needed for each subtask (reading, coding, testing, etc.). Understanding and implementing for the gnowsis framework, as well as,

[5] http://www.w3.org/TR/vcard-rdf

familiarizing with the adapter architecture took him about six hours. Getting around with the IMAP protocol was already done in about three hours.

The student had to realize two types of adapters: He started with implementing a graph based adapter. It took him 24 working hours of implementation plus 16 working hours of debugging. As a second task, a CBD based adapter has to be implemented, too. In contrast to the graph adapter, it took him only about 8 hours to implement the CBD adapter, debugging included.

The CBD adapter was not only easier to implement, but also easier to test (particularly less debugging time): A CBD of some URI can be tested without any RDF API or framework running. Besides, the CBD adapter can be implemented in any language and the RDF/XML representation of the CBD can be created by simple text concatenation. This was exactly the case for the Mozilla address book plugin: an installable extension (XPI-file) integrates some corresponding Java Script code into the native application (Mozilla Firefox) and answers CBD queries via simple XML-RPC calls.

If existing graph implementations can be reused, the cost of developing a wrapping gnowsis adapter is minimal. On 23th of April 2005, Richard Cyganiak created a wrapper for the D2RQ project [5] in about six hours of programming time. The existing D2RQ project is functional and tested, a stable basis for a gnowsis adapter. It was wrapped in a Java project, and the needed glue code has been written, including the formal RDF description of the adapter and a graphical component to configure the adapter.

implementation cost in hours	graph adapter	CBD adapter
familiarize with adapter architecture	2	2
familiarize with gnowsis framework	4	–
reading and testing native data interface	3	3
implementation	24	7
debugging	16	1
deploy adapter	1	1
sum	50	14

Table 2. A comparison of the implementation cost for different adapter types.

6.2 Performance of the adapters

As displayed in table 1 a graph adapter provides full query support (e.g. RDQL), because all triples in the graph are accessible and can be searched very efficiently. In contrast, the CBD adapter provides "jigsaw pieces" around given URIs. Using and combining these, some RDQL queries can be executed, too (searching for subjects is not possible without crawling).

For such RDQL queries the retrieval and combining of the relevant CBDs costs, of course, more time than directly accessing a graph adapter. On the other

hand, if (all) information about a resource shall be retrieved, the CBD provides exactly what is needed, and therefore is as efficient as the graph adapter.

As an example we queried metadata about appointments stored in Microsoft Outlook in form of different queries: (1) the timestamp of one specific appointment, then (2) *all* properties of that appointment, after that (3) the timestamps of 1000 appointments, and finally (4) *all* properties of 1000 appointments. All four queries have been tested on both the graph adapter and the CBD adapter. Table tbl:EvalAppointments shows, that the graph adapter is, of course, faster on the retrieval of a single property, whereas the CBD adapter needs even less time for the retrieval of *all* properties of a resource.

runtime in milli seconds	graph adapter	CBD adapter
one appointment, one property	20-30	20-30
one appointment, all properties	20-30	20-30
1000 appointments, one property	16924	26378
1000 appointments, all properties	30644	26378

Table 3. Comparison of the runtimes of queries performed by the graph adapter and by the CBD adapter.

7 Conclusions and Future plans

The gnowsis adapter framework is published as an open source project and has been used already by both interested individuals and research institutions. Using the presented adapters, it is possible to connect to a variety of data sources and crawl vast RDF graphs. By extracting real-life data sources, gnowsis adapters eventually allow working on big, real RDF data instead of small, irrelevant sample graphs. Using the gnowsis framework and its adapters, programmers can concentrate on the real challenges of the Semantic Web: awakening the Semantic Web by writing and using (tiny, task-specific) applications, using ontologies and inferencing for sophisticated retrieval.

In consequence of the latest developments in the Data Access Working Group of the W3C, and the resulting SPARQL framework, we will concentrate on taking the gnowsis adapter framework in the suggested direction: Each adapter should be represented as a named graph in a graph-set. A single adapter, as well as, the graphset should be published using the SPARQL query language and the according protocol. Then, contacting the framework and relying on this stable protocol, applications can be implemented without any deeper knowledge about adapters or the framework as such. As a part of this strategy, we plan to integrate other SPARQL sources as data sources into the gnowsis framework, resulting in a transparent data flow. GUI applications and semantic web applications will benefit from this development.

References

1. Exif2RDF by Masahide Kanzaki, http://www.kanzaki.com/test/exif2rdf JpegRDF by Norman Walsh http://nwalsh.com/java/jpegrdf/jpegrdf.html.
2. the gnowsis semantic desktop project website http://www.gnowsis.org.
3. Digital still camera image file format standard (exchangeable image file format for digital still cameras: Exif) version 2.1. Technical report, Japan Electronic Industry Development Association (JEIDA), June 1998.
4. Jeen Broekstra, Arjohn Kampman, and Frank van Harmelen. Sesame: A generic architecture for storing and querying rdf and rdf schema. In *Proc. of the International Semantic Web Conference 2002*, 2002.
5. A. Seaborne C. Bizer. D2rq treating non-rdf databases as virtual rdf graphs. In *Proceedings of the 3rd International Semantic Web Conference (ISWC2004)*, 2004.
6. F. Dawson and D. Stenerson. Rfc 2445: Internet calendaring and scheduling core object specification (icalendar), November 1998.
7. Andy Seaborne (edts) Eric Prud'hommeaux. Sparql query language for rdf. W3c working draft, W3C, 2005. http://www.w3.org/TR/rdf-sparql-query/.
8. Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns*. Addison-Wesley, 1995.
9. A. Harth. Seco: mediation services for semantic web data. *Intelligent Systems, IEEE*, Volume 19(Issue 3):66 – 71, May-June 2004.
10. Masahide Kanzaki. Exif vocabulary workspace - rdf schema. Technical report, RDF Interest Group, 2004.
11. Brian McBride. Jena: Implementing the rdf model and syntax specification. In *Proc. of the Semantic Web Workshop WWW2001*, 2001.
12. C. Newman. Rfc 2192: Imap url scheme, September 1997.
13. Leo Sauermann. The gnowsis-using semantic web technologies to build a semantic desktop. Diploma thesis, Technical University of Vienna, 2003.
14. Leo Sauermann and Sven Schwarz. Introducing the gnowsis semantic desktop. In *Proceedings of the International Semantic Web Conference 2004*, 2004.
15. Patrick Stickler. Cbd - concise bounded description. Technical report, NOKIA, 2004. http://sw.nokia.com/uriqa/CBD.html.
16. Patrick Stickler. The uri query agent model - a semantic web enabler. Technical report, NOKIA, 2004. http://sw.nokia.com/uriqa/URIQA.html.