Semantic Web portal – Grounding Technologies and Tools Evaluation

Semantic Web Community Portal Project

Anna V. Zhdanova, Michal Zaremba, Lucas Bischof, Knud Möller

Deliverable:
D6

Version:
0.3

Date:
14-Apr -04

DERI Ireland

University Road

Galway

IRELAND

www.deri.ie

DERI Austria

Technikerstrasse 13

A-6020 Innsbruck

AUSTRIA

www.deri.at

Table of Contents

0 Executive Summary

This document comprises an analysis of existing technologies and tools that can be appropriate for building and maintenance of web-portals of the next generation. I.e., the analysis is a starting point to upgrade existing websites to Semantic Web driven web portals. The case studies, conducted by the SW portal group, i.e., the Semantic Web community portal, the Austrian eTourism portal and People's portal had provided a practical grounding for the conducted here analysis. Thus, the current technologies and tools for portal construction are considered, and the grounding for choosing in favour of certain technologies and tools for Semantic Web portal construction is provided.

April is the cruelest month, breeding
Lilacs out of the dead land, mixing
Memory and desire, stirring
Dull roots with spring rain.

T.S. Eliot, "The Waste Land"

1 Introduction

The aim of this document is to present the evaluation of the selected technologies and tools that may constitute the future building blocks of Semantic Web portals. This assessment is a very selective work as only a very limited number of technologies and tools are evaluated with the manpower resources dedicated to this task. The document is going to grow over time as the evaluation process progress. It is also highly probably that some new technologies will be added to this evaluation, because of the new requirements revealed with the "User Requirements" document. Some changes may also appear during the phase of the portal implementation.

Since version 0.2 of this document the portal architecture is explained. A list of the suggested technologies and tools fulfilling requirements of each of the architecture tier, which should be evaluated in the coming month with the names of the researchers responsible for carrying evaluation, has been provided. New technologies and tools may be added and some may be removed from this report while carrying out the evaluation work.

Each of the below mentioned technologies and tools could be itself a good material for the whole book, but the aim of the authors is to have their work practical and this document short and concise. The authors attempt to carry a comprehensive evaluation, not only limited to reading FAQs or white papers, but to the real testing based on the initial implementation of the selected features (components) of the portal. Conclusions from the authors' work and advice regarding applicability of given technologies for the Semantic Web portal are going to constitute the main part of this document. It has been also the intention of the authors to spend as much time as possible on evaluation and prototyping of various features of the portal with given technologies or tools and not on

the production of the high level overviews that can be easily found anywhere. Links are provided to forward a reader for further reading.

The document is structured as follows. 3-tier and N-tier architecture are presented in the section 2 and explanation why the 3-tier architecture has been chosen for the procedure of technologies evaluation is clarified. Section 3 dwells on a problem of creation of next-generation applications, here Semantic Web portals, with current technologies and tools. Section 4 and 5 provide the set of current technologies and supporting tools that are evaluated with respect to their applicability to development of Semantic Web portals. Given technologies and tools are assigned to particular tiers of the 3-tier architecture.

2 Architecture

During the course of evaluation of the technologies and tools suggested for the development of the Semantic Web portals, the assumption is to be made that the portal is going to have the 3-tier architecture (see Figure 1). The authors are trying to address the requirements of the presentation, application logic and resource management layers from the 3-tier architecture by evaluating technologies and tools that may constitute the future building blocks for each of these tiers.

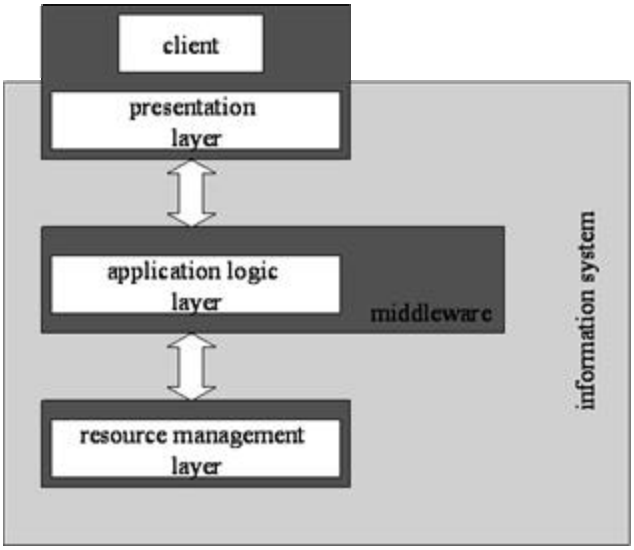

Figure 1: Three tier architecture [Alonso03]

The layers (tiers) are defined as follow:

- **Presentation layer** is responsible for the communication with the external entities as for example humans or other systems. For some types of Semantic Web portals (e.g., B2B) these other systems will be mainly other semantic web enabled portals.
- **Application logic layer** is responsible for the processing of data before the results are ready to be delivered to the presentation layer. Application logic layer exposes services offered by the Semantic Web portal.

- **Resource management layer** includes all the elements of the semanticweb.org portal, which are capable to preserve data as for example databases, repositories, file systems etc.

The reader should refer to Alonso et al. [Alonso03] to understand the complexity and advantages of the various types of tier architectures and the purpose of each of the layer in the 3-tier architecture.

While there is no harm to assume that the portal represents 3-tier architecture at the time while evaluating the technologies and tools, it can be easily predicted that the "User Requirements" analysis for the semanticweb.org portal will require the N-tier architecture to be used for the portal implementation (see Figure 2).

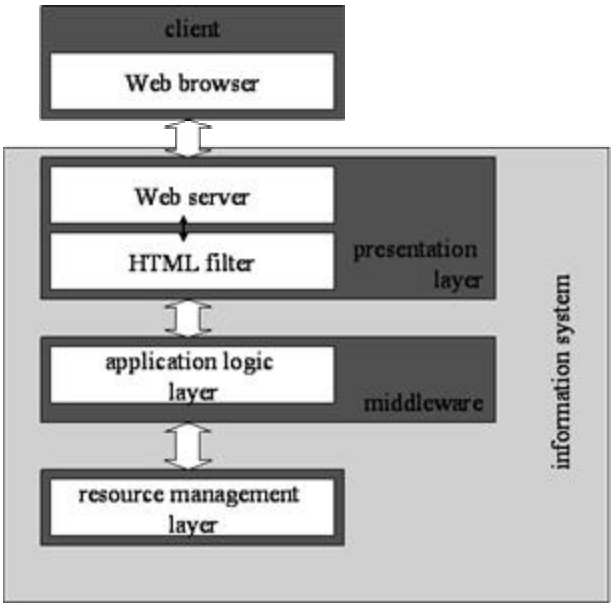

Figure 1: An N-tier system [Alonso03]

As presented by Alonso et al. [Alonso03] the N-tier architecture is the result of applying the 3-tier model in its full generality and increased relevance of the Internet as an access channel. It is almost sure that the architecture of the Semantic Web portals will

represent the web application architecture, which in its definition is the N-tier architecture. Also the potential requirement to enable the exchange of data with other semantic web portals (or saying more general with other information systems) indicates the necessity of the application logic layer or the resource management layer of the semanticweb.org portal to be able to communicate with the presentations layers of other portals. This automatically indicates that the N-tier architecture should be used.

J2EE (Java 2 Enterprise Edition) architecture is the example of the application of the N-tier model (the reader should notice that the authors do not refer here to particular J2EE implementation, but to architecture that has been developed with the Java Community Process [Java]). The decision to select J2EE architecture is rather pragmatic than fully rational. There is a very competitive N-tier architecture offered by Microsoft with the .Net platform. Having limited number of manpower resources and already good familiarity of the J2EE platform, the authors decided not to sacrifice a couple of months only to understand the basics of .Net, but to go straight with the technology that is already partially well-known to them.

3 Gap between Specification, Support and Applications

The next generation application to be strived for is a Semantic Web portal, i.e., a web portal which is based on Semantic Web technologies. In turn, a web portal is a web site that collects information for a group of users that have common interests [Heflin03]. Thus, researchers and developers nowadays face the problem of making semantically friendly technologies and tools work for the existing web portals. That means that the purpose of the next-generation portals will remain the same as the purpose of the preceding web-portals, however, the main issue is to apply and evolve existing technologies and tools in order to provide full support to the new portal applications.

Semantic Web portals can be seen as an application of Semantic Web technologies and tools, while these semantically friendly technologies and tools are a guarantor of support for any Semantic Web applications including Semantic Web portals. Further, understanding of a problem and specification of its solution is essential for the problem resolution. Specifically here, a problem of bringing semantics to existing web portals is considered, to be precise, a step towards new applications is desired to be taken. Therefore, in order to have new applications developed, a support in terms of technologies and tools is to be provided. Theoretically, at first a problem is recognized, than its solution is specified, then the means/support to implement the solution are chosen, and at last, a new application is brought into reality. Then, the reality generates new problems and at once makes obsolete all the developed so far specifications, support and applications. The cycle of this process of moving from one specification-support-application state to another is depicted at Figure 2.

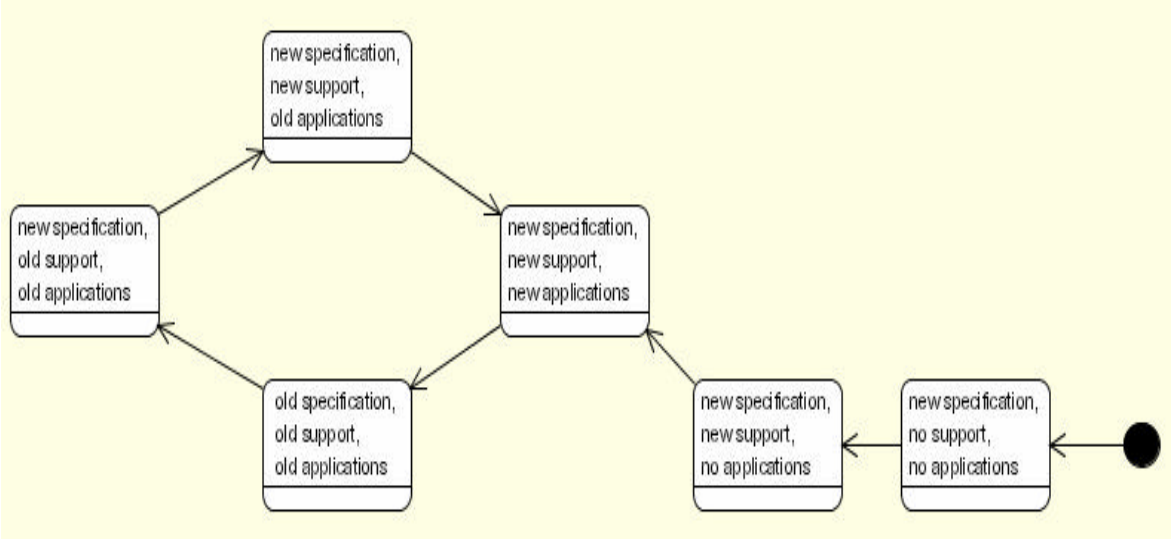

Figure 2: Cycle of emerging of new applications

However, in user-driven practice, only new real-life applications do count, since only they provide useful means/support to the final consumer. Thus, it is natural to attempt to make a direct transition from the state of new specification, old support and old applications to the state of new specification, new support and new applications. But this transition can not be direct, because the state of acquiring new support has to take place in between. Here, this general problem of gap between new specification, old support and new desired applications concerns also the future Semantic Web portal applications. Nowadays, the Semantic Web is supported by tools that are primarily the products of traditional AI research and have been transitioned on the World Wide Web (e.g., Protégé, OILEd, OntoEdit). As such, these tools can be very powerful, but they only focus on some parts of the "lifecycle" of Semantic Web content [Golbeck02]. Thus, for each type of Semantic Web solution, such as implementation Semantic Web portal application, available technologies and tools are to be considered with respect to their applicability to the new problem, and; if they are not sufficient, the new technologies and tools are to be specified and developed.

The following sections on technologies and tools (Sections 4 and 5) will contain a view on the acuteness of the problem of gap between a specific technology or tool and the desired new support for the next-generation web portal. The considered specific technologies and tools are seen by the authors as the most appropriate among the existing technologies or tools for the Semantic Web portal implementation and are chosen with respect to the guidelines listed in the Introduction.

4 Supporting Technologies

This section is structured around three layers of the 3-tier architecture, but the reader should remember, that finally the N-tier architecture is going to be used to build the Semantic Web portal.

This very selective list should be extended together with the increase of the experience and the practical skills of the authors of this overview. At this stage the technologies are analysed in separation, and this analysis gives a feedback to drawing general conclusions.

4.1 Presentation Layer

4.1.1 HTML, JavaScript and Their Variations

Despite the multitude of Web languages, HTML (tag language) and JavaScript (script language) are still the most frequently used and known ones. Among the most important features of HTML are: displaying tables, displaying layer, text formatting, creating links, input of graphical objects, displaying formulary fields. Among the important variations and extensions of HTML are: XHTML is a reformulation of HTML 4 as an application of XML, the StyleSheet (CSS, XSL) is a technology for design specification and import, Dynamic HTML (DHTML) is an enabler of interactivity features in HTML pages. JavaScript was invented by Netscape, and Jscript is JavaScript's equivalent developed by Microsoft. JavaScript is supported by the Document Object Model (DOM) that is a recommendation by W3C. The basic idea of JavaScript was to fit in static HTML web pages the dynamic possibilities given by an event handler (e.g., OnMouseOver, OnSubmit,…) and the object orientation (e.g., document.Formular.Input.value = …).

HTML, JavaScript and their variations are essential for most of the human-computer interaction web-portal scenarios. Generally speaking, these languages are today's corner stones for data presentation to a human and also for human-computer data exchange of

certain types. Actually, these languages are essential due to the fact that only specifications of these languages nowadays have a strong support by existing popular web browsers. Trying to use different presentation technologies in Semantic Web portals nowadays will likely cause an impassable gap between specifications and support (see Section 3). However, most of the dynamic functionalities of JavaScript or DHTML can be substituted by the JSP and Java servlet functionalities, though such substitution is not always computationally cheaper.

4.1.2 Java Server Pages – JSP

JavaServer Pages (JSP) technology provides a simplified, fast way to create dynamic web content. JSP technology enables rapid development of web-based applications that are server- and platform-independent.

JSP technology uses XML-like tags that encapsulate the logic that generates the content for the page. The application logic can reside in server-based resources (such as JavaBeans component architecture) that the page accesses with these tags. Any and all formatting (HTML or XML) tags are passed directly back to the response page. By separating the page logic from its design and display and supporting a reusable component-based design, JSP technology makes it faster and easier than ever to build Web-based applications.

Together, JSP technology and servlets provide an attractive alternative to other types of dynamic Web scripting/programming by offering: platform independence; enhanced performance; separation of logic from display; ease of administration; extensibility into the enterprise and ease of use.

It should be noted that the use of JSPs is always optional – one can generate the same JSPs' effects with Java servlets at an expense of mixing content with layout. However, if an application is a small one, using servlets only without JSPs can be worthwhile. Similar to HTML forms and servlets' handling of parameters, the future shape of JSP technology depends on whether a human will be capable of communication with Semantic Web portals in semantic languages, but not in parameter-value pairs as it is now.

4.2 Application Logic Layer

4.2.1 Servlets

A servlet is a small Java program that runs within a Web server. Servlets receive and respond to requests from Web clients, usually across HTTP, the HyperText Transfer Protocol. The relevant for the Semantic Web portals servlet interface is HttpServlet that allows to create an HTTP servlet suitable for a Web site. A subclass of `HttpServlet` must override at least one method, usually one of these:

- `doGet`, if the servlet supports HTTP GET requests
- `doPost`, for HTTP POST requests
- `doPut`, for HTTP PUT requests
- `doDelete`, for HTTP DELETE requests
- `init` and `destroy`, to manage resources that are held for the life of the servlet
- `getServletInfo`, which the servlet uses to provide information about itself

Servlets typically run on multithreaded servers, thus servlets must be designed to handle concurrent requests and to synchronize access to shared resources. Shared resources include in-memory data such as instance or class variables and external objects such as files, database connections, and network connections. Indeed, any Semantic Web portal is a typical case in terms of concurrency, since such a portal presumes having multiple users sharing the same portal resources.

The existing servlet technology is rather simple, but robust. The functionality provided by this technology is tightly interrelated with presentation level on one side, since servlets take an input as parameters from forms, embedded in JSP and HTML files. On the other side, the servlet technology is designed to work with JDK's well-developed access to traditional resource management layer technology, such as relational databases accessed via the Statement interface for executing a static SQL statement and returning the results it produces.

It should be noticed that nowadays popular technologies at presentation or resource management layer can not be considered to be semantic. E.g., the internet forms are not annotated with any semantic data, and when a servlet takes provided by the user parameters from such a form, the servlet has to refer to an oracle on how to handle these parameters. Also, relational databases do not bear much semantics, so the servlet has to communicate with an oracle again concerning the database treatment. Clearly, the changes of the storage and presentation technologies will cause the changes in the servlet behavior. However, since servlets are merely Java programs, in case of change in protocols, presentation or resource management layers, an adaptation to the new,

Semantic Web environment would be possible and reasonable. We believe that the idea in the base of the servlet technology will remain invariant for any possible future Web applications: there is a need to receive and respond to requests from Web clients.

4.2.2 Enterprise Java Beans – EJB (???)

4.2.3 OWL /RDF / RDFS / XML APIs, Query Interfaces

Most of the existing APIs for manipulation of data encoded in RDF, OWL and other W3C ontology languages have naturally derived or organized in a similar way as Apache's Xerces for XML.

The examples for query interfaces are RDQL - a query language for Jena, SNOBASE[1] interfaces that tend to unify interfaces for the W3C standardized ontology languages under the same interface. The problem with evaluation of query interfaces is that the practical wide use of the existing few applicable to real world query processors is still very rare.

4.3 Resource Management Layer

4.3.1 Repositories/Ontology-Repositories/Database Systems

With usual web-languages (HTML, CSS, JavaScript) you are not able to create or store any ontology. To create an ontology online, a server-sided program-language is very useful. The three most important ones are CGI/Perl, ASP and PHP. Furthermore ActiveX, Java or C are possible languages to program web applications in this area. For young programmer ASP or components for ActiveX are normally uninteresting because of the licence expense. The same problem exists for Java if someone produces commercial software/application. CGI/Perl and PHP are very similar in programming, but PHP has got a big community of interested programmer behind, who are interested in creating newer and more comfortable versions.

[1] IBM's SNOBASE: http://www.alphaworks.ibm.com/tech/snobase

The technology behind these languages is more or less the same: A specific request is send to the server. This request is created out of an HTML page. The server calculates the input (storing in databases, storing in files, opening files, creating files, calculating…) and gives out to the user browser an HTML/XHTML file with the result. That means server sided. A normal web page is client sided, because the client browser has to make calculations to present the HTML/XHTML/XML file.

Furthermore ontology repositories can be created with the help of client-server applications. A program, written in Java, C, C++, VisualBasic, Borland Delphi, or others is situated on a server. The user has got a smaller program, mostly in the same language and sends via internet input to the server program. The server program handles the data and stores them for example in a database.

Software systems can do the same, but normally you have to start the program on the server, where the data will be stored. Such a system might be uninteresting if many people or systems work on the same project via internet.

One big problem of programming database applications supporting ontologies is the performance. The storing of for example more than 50 000 data in a logic and dynamic way available via internet can strike down any server above all if there are many people sending request at the same time.

To increase performance there are two ways. One is to use the database only for storing information and to create a XML-based output, stored in an intelligent way to find information quick and easy. The user does not have to search in the database, but in the created XML files via for example XPath[2]. The other possibility is to create another database system, a database system which is more intelligent and therefore quicker than the typical ones.

Following the second way, SynapticWorkgroup[3] tries to develop a system called Dynamic. The base idea is to rewrite a database access and to create an access which

[2] http://www.w3c.org

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

allows searching and writing information in more than one database "cell". An example: ontology for persons (name, address, hobbies). The hobbies of a person might be a big problem. You have got one database table for the hobbies one for the persons and one to assign a hobby to a person. If you have got 50 000 persons and each around 10 hobbies, the assigned table has got around 10 * 50 000 entries. To give out the hobbies for a person, a normal database system has to search in 500 000 hobby-entries to find the applied ones. In Dynamic you have to search the correct persons in the 50 000 entries only. The only problem is that one person is not allowed to have more than one hundred hobbies.

To conclude, the possibilities to save ontologies to share them with other systems and users are as follows:

- Input → Platform → typical Database;
  Output → HTML/XML/RDF/PDF… (usual online shops and other web based platforms)
- Input → Platform → modified Database;
  Output → HTML/XML/RDF/PDF… (Dynamic from SynapticWorkgroup)
- Input → Platform → files;
  Output → XML – Files
- Input → Program File;
  Output → HTML/XHTML – Files (DELTA System)
- Input → Program File;
  Output → XML/RDF/RDF Schema (some XML editors; i.e. Protégé)
- Input → Program File;
  Output → DAML+OIL/OWL + XML/RDF/RDF Schema (some XML editors; i.e. Protégé)
- Input → Program File → database;
  Output → to what ever from a database

4.3.2 CMS Systems (???)

4.3.3 Blogs, Wikis (Laurentiu, Ina)

5 Supporting Tools

Tools that are capable to work with the technologies discussed in section 4 are presented below. Again names of people carrying initial implementation and testing of the portal functionality with the given tools have been included in brackets.

The selection of tools has been based on several criteria. The tools must be open and free source. They need recognition by the community of software developers. They must be J2EE compatible or there must be some ways to use them with Java, as architecture of the Semantic Web portal is going to be built on Java "standard".

5.1 Presentation Layer

5.1.1 Tools for Presentation and Content Management Level Development Support

After a platform was created, the work is not finished. The work goes on, actually. A platform should be dynamic in design and in content. Information changes in time, and there should be users who are able and interested in updating all the data. Many different systems were created and many more will be created in future to handle the problem of updating.

Updating a page in content and design needs affords in knowing the language in which the content was written or the help of an intelligent application. One possibility is to take the source code and make changing where it is necessary. To change the source code one needs time and the ability to do that. Although HTML is not a very complicated language, only a limited group people have learned to handle the source code. Therefore, one very important point in programming a platform or similar is how and by whom the content should be updated and changed. The user spending time in updating the content should get an easy tool, to handle the process of updating as easy as possible. The user should not be required to learn any web language to do his work.

All the many Wiki derivates in any program language do this in an simple way: everyone is allowed to edit content. Wiki tells the users not how to manage the content. They can do it in this way which seems to be the best for them. A system like Wiki is very interesting for sharing information and thoughts to the community.

CMSs (Content Management Systems) try to handle information with care. A security system should only allow a selected group of users to work on the content. Straight rules are implemented to handle who is allowed to make somewhere some changes. The problem is that a CMS might be much more complicated to implement, but it guarantees strict data handling (especially important for ontology based systems). A user is authenticated by a login. The authentication is important if an incorrect input may lead to a loss of money or similar.

There are many different software systems which integrate the production and the content management of a platform. There are WYSIWYG Editors like FrontPage (Microsoft) or Dreamweaver (Macromedia), there are different Wiki tools, ColdFusion (based on Wiki), software systems like Interwoven or many specially created systems to handle the information for one specific platform. Some of them have tools to cope with XML, and some not. But no one was found to handle especially ontology based content. Although, this might be the future.

5.2 Application Logic Layer

5.2.1 Manipulating RDF and Reasoning – Jena

Nowadays, Jena can be considered to be the main Java library for RDF and other W3C ontology languages and reasoning over data described in them. Since Jena is typical java library, it is easy to integrate in any Java application. One can say that Jena upgraded XML an XML support to a full-fledged RDF support. Practically, Jena includes a Xerces implementation and probably heavily depends on it.

The basic functions to manipulate RDF such as import, export, parsing and operation on RDF models (e.g., merging, difference) are implemented in Jena. With appearance of new ontology languages such as DAML and OWL, the enthusiasm of the Jena developers looked as if monotonously decreasing. Indeed, providing a full-fledged

support including properly (re)named interfaces to the ever-emerging ontology languages can be viewed as an unrewarding task.

The following descriptions can be found regarding reasoning in supported by Jena ontology languages such as RDF(S), DAML and OWL.

RDF(S) reasoning: "A full implementation of RDFS reasoning using a hybrid rule system, together with optimized subclass/subproperty closure using the transitive graph caches. Implements the container membership property rules using an optional data scanning hook. Implements datatype range validation."

DAML reasoning: "We do not support DAML inference. This is a slightly extended version of the RDFS reasoner to support some interesting subsets of DAML that correspond roughly to what was there in Jena1."

OWL reasoning: "A pure forward chaining implementation of the experimental OWL closure rules based upon the basic forward rule interpreter. This is not a serious or usable OWL implementation, it is a tool for developing the rules."

Thus, RDF(S) remains to be the most supported ontology language in Jena. RDQL is an implementation of an SQL-like query language in Jena for RDF(S). RDF there is regarded as triple data, without schema or ontology information unless explicitly included in the RDF source. RDQL provides a query syntax and query execution engine in Java, primarily for experimental and embedded applications.

Still, the acute need to handle a variety of similar languages (specifically, RDFS, DAML and the OWLs), and to bring them all to work well with Jena 2s new inference capabilities, the ontology API in Jena was designed. Whereas in the old DAML API, the Java classes were tightly bound to the language being processed (e.g. DAMLClass, DAMLObjectProperty, etc.), the ontology API is language-neutral (thus the classes are now OntClass and ObjectProperty). To support this, each of the languages has a *profile*, which lists the permitted constructs and the URI's of the classes and properties.

Thus, here we have the case when the creators of a support tool (Jena) had to create an own specification (ontology API) to handle support of other specifications (RDF, DML, OWL).

5.2.2 JSPs and Servlets Container – Tomcat

Clearly, after the interactive part of a Semantic Web portal application is developed in form of JSPs and servlets, it needs to be stored and provided with a possibility to have an outreach to the Web client. These two functions are provided by Apache's Tomcat server. Tomcat supports well applications of any complexity developed in Java and does not really depend on whether these applications process semantic data or ordinary data.

5.2.3 EJB container – JBoss (???)

5.2.4 RDF store – Sesame (???)

5.3 Resource Management Layer

5.3.1 CMS – Open CMS (???)

5.3.2 Blogs – HP Blogs (Laurentiu, Ina)

5.3.3 Ontology Server – Protégé server - (???)

5.3.4 Database System – Postgresql, MySQL

Databases might be a very practical way to store an ontology and to give out XML-files or similar for further information sharing.

Today there are many databases available, the most well-known ones are:

- Oracle (Oracle9i)
- mSQL
- MySQL
- IBM DB2
- Borland JDataStore
- Borland Interbase
- Adabas D
- SYBASE ASE
- IDS Server
- Postgresql

Because not every database works together with every program language in the same surrounding (i.e. Linux, Windows, Macintosh …). The combination might be very important for the best performance. Well-working combinations are known to be as follows:

Linux → Apache (Server) → PHP → MySQL → XML output

Microsoft → InternetInformationServer (IIS) (Server) → ASP → mSQL → XML output

Any operating system → Java Web Start (Client) → Java → Oracle XML output

Presenting data in the internet the decision to create a search via access to the database or via access to created HTML/Text/XML files might be very important in view of the performance of the system.

When an open source solution is required, PostgreSQL or MySQL are to be taken. PostgreSQL is an object-relational database management system (ORDBMS) based on POSTGRES, Version 4.2, developed at the University of California at Berkeley Computer Science Department. PostgreSQL is an open-source descendant of this original Berkeley code. It supports SQL92 and SQL99 and offers additional features such as: complex queries, foreign keys, triggers, views, transactional integrity, multiversion concurrency control. Also, PostgreSQL can be extended by the user in many ways, for example by adding new data types, functions, operators, aggregate functions, index methods, procedural languages. And because of the liberal license, PostgreSQL can be used, modified, and distributed by everyone free of charge for any purpose, be it private, commercial, or academic. MySQL can be an alternative for an open source solution.

Some basic ideas that lie in the base of databases can be seen as undermining Semantic Web approach. The fact that each content provider organizes their data in their database in their own way and does not semantically annotate it to be accessible and manageable by other content providers and services makes integration very difficult. If the portals are automatically generated by the same system (e.g., as described by Corcho et al. [Corcho03]) simplifies integration, but one can not expect that all the portals will be generated by the same system. An example of how a high interoperability was practically reached for multiple heterogeneous content providers by introduction of an RDF layer on top of heterogeneous databases is described by Hyvönen et al. [Hyvönen04]. Clearly, effective storage and access to ontological data is still an open issue: one would like to combine the high performance of developed databases with ubiquity, homogeneity and semantic power of knowledge represented in popular ontology languages.

5.4 Development Assisting Tools

5.4.1 Project build tool – Ant

Ant[4] (Another Neat Tool) is a build tool similar to tools like Make or Jam. It is an open source project[5] maintained by the Apache Software Foundation. Ant started out as part of the Tomcat project, but - due to its increasing popularity - turned into an independent project in July 2000. Since then, it has spread rapidly and is now the build tool of choice for most Java developers. As of the time of writing of this document (March 2004), the project's latest stable version is 1.6.

In general, build tools have the purpose to automate various non-creative and time-consuming processes in software development, such as the compilation of code, archiving of compiled code, bundling and structuring of resources, deployment of software for different targets and platforms, etc. The different processes will have to be formulated in a build script. Each time the code base of the software project has changed significantly and a new version has to be tested or deployed, the script can be executed to perform the tasks described above. The obvious advantage of using build tools in general is that they can save a lot of time and minimize the amount of errors in the deployment process. Furthermore, build tools can be useful to increase consistency in collaborative development, as they can be used to define build styles for an organization, company or developer team.

Ant has a number of characteristics which make if different from other build tools. These characteristics are at the same time Ant's main advantages:

- Ant is implemented in Java and therefore platform independent. If needed, Ant's functionality can easily be extended using Java classes.
- Ant's build scripts are written in XML instead of some obscure and cryptic other language. As XML is widespread and easily readable, the learning curve for Ant is less steep. Using XML also makes writing scripts less error-prone.

Even though Ant's expressive power might be somewhat smaller than that of e.g. shell script based tools, this doesn't necessarily pose a big problem. Whenever a shell command is absolutely needed, this command can be executed by falling back on Ant's `<exec>` command.

---

[4] ANT: http://ant.apache.org

[5] The Apache Software License Version 2.0: http://ant.apache.org/license.html

Finally, Ant can be easily integrated into many Integrated Development Environments (IDE)[6]. For our purposes it is mainly interesting that Ant is fully supported by the Eclipse IDE, since this development environment is mainly used by DERI developers.

5.4.2 Ontology Editors - Protégé

For certain portal types, for instance, for B2C portals, a portal ontology is naturally developed by the business side, and accessed by the customer side. Since in certain B2C portals a portal ontology is not expected to be populated, developed and modified by many parties, ontology evolution these B2C cases nowadays can be supported by existing ontology editors. Observing functionality of exiting tools that are positioned as ontology editors by their developers, one can duly define an ontology editor as a tool allowing the user to build ontologies (via graphical interface in most cases) and import/export ontological descriptions in certain supported ontology languages.

The number of existing ontology editors is relatively large. However, most of the existing ontology editors are inconvenient to use or are preferred to another better editor when the time to choose comes. Here, we will briefly discuss OilEd[7] (an ontology editor allowing the user to build ontologies using DAML+OIL, developed in the University of Manchester) and OntoEdit[8] (an Ontology Engineering Environment, developed in Ontoprise GMBH, Karlsruhe). It is doubtful whether an ontology editor should provide an intensive support for reasoning like OilEd (with Fact reasoner), or include strong visualization tools like OntoEdit. Indeed, when reasoning is needed in an application, testing of the querying module is performed on the application's reasoner (but not on the editor's one), and the testing module is organized to run subsequently multiple queries, deliver statistics, and thus outrank any testing that can be done in an ontology editor. Existing ontology editors that concentrate on extensive query support can be rather considered to be prototypes that demonstrate the power of ontology technology, than tools of real practical use. Another example of a discrepant functionality is

[6] Ant with IDE: http://ant.apache.org/external.html#IDE%20and%20Editor%20Integration

[7] OilEd: http://oiled.man.ac.uk

[8] Ontoprise's OntoEdit: http://www.ontoprise.de

ontology visualization: if the users find convenient to browse through the folder structures since long time ago, the visualization function of the ontology "folder" structure in an ontology editor can not be expected to be truly helpful. Otherwise, the visualisation modules would have to be developed for the folder structures in general. Although, approaches to include visualization or reasoning may bring innovation, concentrating on these approaches might not be the best idea if the goal is to build an ontology editor that will be popular here and now having quite limited resources. Such directions as concentration on providing support for permanently emerging acknowledged ontology formalisms, integration/development of companion technologies, improving system's performance and user interfaces look to be more reasonable.

At the moment, maybe the most successful ontology editor is Protégé[9]. Protégé is a well-developed (also in Java) ontology editor that supports W3C standardized ontology languages, such as RDF and OWL in constructing and population domain ontologies. Also, Protégé is a platform which can be extended in the user-convenient way and it is a library that can be used by other applications and tools. These Protégé's features made the tool flexible, easy to integrate and thus popular. The Protégé research team is active also in developing other tools for the Semantic Web applications, such as PROMPT for ontology merging and mapping [Noy03]. Development of such related tools means being involved in other important technology trends that are likely to modify ontology editors in general, and thus gaining more chances to survive by integrating these tools with the developed ontology editor. However, Protégé has also the reverse side of its generally acknowledged advantages. For instance, the tool allows installing multiple plugins, but does not guarantee peaceful coexistence of these plugins with each other. E.g., the OWL plugin for Protégé conflicts with the RDF plugin for Protégé in case they both appear in configuration of the same instance of installed tool.

5.4.3 Project IDE (Integrated Development Environment) – Eclipse

An Integrated Development Environments (IDE) is a tool to aid software development. Even though software development can theoretically be done using

[9] Protégé: http://protege.stanford.edu

nothing but a command line compiler and a simple text editor, IDEs can speed up the development process significantly by supporting or automating many of the typical programming tasks through a graphical user interface, from basic things such as code editing to more complex undertakings like project management and deployment. Features found in many IDEs are integrated compiling, linking and building, wizards for different kinds of projects, automatic code completion and formatting, automatic generation of code stubs, syntax highlighting and checking, graphical GUI-building, etc..

Eclipse is a free, open source and Java-based IDE, maintained by a board of stewards founded in November 2001. The board includes members from a number of large companies, among them IBM, Intel, SAP and Hewlett Packard. Even though not necessarily a guarantee, this does suggest that there will be good maintenance and further development of the Eclipse platform in the foreseeable future. As of the time of writing of this document (April 2004), the latest stable build is 3.0 Milestone 8, which is distributed under the Eclipse Public License [10].

While Java-based and therefore platform-independent in principle, the Eclipse platform also makes heavy use of the SWT (Standard Widget Toolkit) user interface library. Instead of the lightweight nature of Sun's own Swing library, the SWT follows a heavyweight approach in rendering the individual UI components, and therefore has to rely on a native implementation for each individual platform. While this somewhat compromises platform independence, there are ports to virtually all major development platforms, incl. Linux, Solaris 8, AIX, HP-UX, Mac OSX and all versions of MS-Windows since 98. For a complete list of ports, see http://www.eclipse.org/eclipse/development/eclipse_project_plan_3_0.html .

Although Eclipse is often conceived as an IDE solely for Java development, this is actually not correct. Rather, Eclipse is a general purpose IDE that provides some basic functionality and a framework to extend it for various purposes, using plugins. While the Java plugin (JDT - Java Development Tooling) is included in the main Eclipse distribution, there is actually a great variety of other plugins available from 3rd party developers, including IDEs for programming languages such as Pearl, C#, PHP or Ruby, a WSDL viewer, a data-base access plugin, a Subversion plugin, a Google search plugin and even a number of games (to showcase Eclipse's extensibility). A (non-exhaustive) list of other plugins can be found at http://www.eclipse.org/community/index.html . This extensibility, together with its open source character, wide acceptance and strong industry support make the Eclipse platform a good choice for developing editor plugins for DERI's own standardization efforts, e.g. WSMO/WSML.

Using the JDT, the Eclipse platform is turned into a fully-fledged Java IDE. It includes extensive project management capabilities such as version control (CVS comes

[10] Eclipse Public Licence: http://www.eclipse.org/org/documents/epl-v10.html

out of the box, other version control systems can be added via additional plugins) and ANT support, a hierarchical class browser, state-of-the-art code editing (including syntax coloring, code generation, formatting and completion), excellent refactoring capabilities, debugging, incremental compilation (errors in the code will be higlighted while typing) and other features. For a complete list of the current features, see http://download.eclipse.org/downloads/drops/S-3.0M8-200403261517/ . It is important to notice that this vast amount of functionality requires an equally vast amount of processing power.

Admittedly, many if not most of the features of the Eclipse platform and the JDT can be found in many other Java IDEs as well. However, a number of aspects make the Eclipse platform stand out. Eclipse is:

- highly extensible
- open source
- platform-independent (to a certain degree)
- free of charge
- supported by a large community of developers and industrial partners

6 Conclusions

In this paper, the existing technologies and tools that can be helpful for creation of a full-fledged Semantic Web portal were evaluated. The selection and analysis of the suitable tools is an interesting process from the research point of view, because, as it was described in the paper, the problem of transition to the next-generation applications (here, the Semantic Web portals) basing on the "this-generation" technologies and tools (analyzed here) was addressed for every aspect of the Semantic Web portal construction.

Among the conclusions, at first, it has to be admitted that the problem of gap between applications, tools and technologies does not have the same degree of acuteness for every aspect of the Semantic Web portal construction. For example, the choice of the project build tool, the project IDE, the servlet type, the servlet container and the presentation layer technologies and tools is expected to be the same as for an ordinary web-portal. The reason for this is that certain technologies and tools lie beyond semantics and knowledge representation, and thus unaffected by addition of the semantic to the portal.

The second conclusion point is that full potential does not imply fertility. Clarification: the movement towards the Web of the Semantic Web portals brings many intermediate specification efforts that are not ultimately implemented in the next-generation applications disregarding even possible existence of the supporting technologies and tools. The Semantic Web field is rapidly developing and many specifications and standards are being observed to appear. However, most of these specifications and standards prove to be the "dead branches of evolution" in a sense that the next-generation applications happen to be based on something else, but not on these specifications. It has been shown, that even a comprehensive and accessible support for a specific application does not guarantee the fertility of this specification. For instance, many DAML supporting features were implemented in Jena, but they are unlikely to be used in the future applications due to the DAML's transformation into DAML+OIL and then OWL. In order to be safe from adaptation to rapid name change of ontology languages and other marketing fluctuations, a tendency to create and employ unified, general interfaces is shown to be prevailing among the developers of supporting technologies and tools (e.g., SNOBASE, Jena).

At third, it has been clearly shown that to overcome the specification-application gap, new technologies and tools (e.g., ontology editors and ontology repositories) appear and get employed in the construction of the next-generation web-portals. Meanwhile, some other existing technologies and tools (e.g., certain type of databases) lose their importance when switching to the Semantic Web portals.

Acknowledgements

The authors thank members of the DERI Semantic Web portal work group (http://www.nextwebgeneration.org/projects/sw-portal/sw-portal.htm) for creation of a stimulating research environment.

References

[Alonso03] Alonso, G., Casati, F., Kuno, H., Machiraju, V., 2003. Web Services -- Concepts, Architectures and Applications, Springer Verlag.

[Broekstra02] Broekstra, J., Kampman, A., van Harmelen, F., 2002. Sesame: A Generic Architecture for Storing and Querying RDF and RDF Schema. In: Ian Horrocks and James Hendler (Eds.), Proceedings of the First International Semantic Web Conference, Springer, LNCS 2342, pp. 54-68.

[Corcho03] Corcho, O., Gomez-Perez, A., Lopez-Cima, A., Lopez-Garcia, V., Suarez-Figueroa, M., 2003. ODESeW. Automatic Generation of Knowledge Portals for Intranets and Extranets. In: Fensel, D. et al. (Eds.), Proceedings of the Second International Semantic Web Conference; Springer, LNCS 2870, pp. 802-817.

[Java] JavaTM 2 Platform, Enterprise Edition 1.4 (J2EE 1.4) Specification. http://www.jcp.org/en/jsr/detail?id=151

[Jena] Jena -- a Java framework for writing Semantic Web applications. http://jena.sourceforge.net

[Golbeck02] Golbeck, J., Grove, M., Parsia, B., Kalyanpur, A., Hendler, J., 2002. New Tools for the Semantic Web In: Asunci{\'o}n G{\'o}mez-P{\'e}rez and V. Richard Benjamins (Eds.), Knowledge Engineering and Knowledge Management. Ontologies

and the Semantic Web, 13th International Conference, EKAW 2002, Siguenza, Spain, October 1-4, 2002, Proceedings. LNCS 2473 Springer, pp. 392-400.

[Heflin03] Heflin J., 2003. Web Ontology Language (OWL) Use Cases and Requirements. W3C Working Draft, 31 March 2003.

[Hyvönen04] Hyvönen. E., Junnila, M., Kettula, S., Mäkelä, S., Saarela, S., Salminen, M., Syreeni, A., Valo, A., and Viljanen, K., 2004. MuseumFinland — Finnish Museums on the Semantic Web: User's Perspective. Proceedings of Museums and the Web 2004 (MW2004), Arlington, Virginia, USA, March 31 – April 3, 2004.

[Noy03] Noy, N.F., Musen, M.A., 2003. The PROMPT suite: interactive tools for ontology merging and mapping. Int. J. Human-Computer Studies 59, pp. 983-1024.