An Architecture for a Semantic Portal

Gerald Reif and Harald Gall

University Zurich, Department of Informatics, 8057 Zurich, Switzerland
{reif,gall}@ifi.unizh.ch,
WWW home page: http://seal.ifi.unizh.ch/

Abstract. Current Web applications provide their information and functionalities to human users only. To make Web applications also accessible for machines, the Semantic Web proposes an extension of the current Web, that describes the semantics of the content and the services explicitly with machine-processable meta-data. In this paper we introduce an architecture of a *Semantic Portal* that provides a unique front-end to the information and functionalities of individual Semantic Web applications. To realize the portal we use WEESA to semantically annotate Web applications and provide the annotations in a knowledge base (KB) for download and querying. Based on that, the Semantic Harvester collects the KBs from individual Semantic Web applications to build the global KB of the Semantic Portal. Finally, we use Semantic Web services to make the portal a unique interface to the services of the Web applications.

1 Introduction

Traditional Web applications serve two main purposes. First, to provide the information presented on Web pages to the user and second as user interface to a back-end software application such as a shopping application or a Web mail client. In both cases humans are the ones who consume the information or operate the application. Machines do not have access to the information and the functionalities.

For this reason, the Semantic Web introduces an extension of the current Web to make the content and the functionality accessible to machines. To make the content machine-accessible, Web pages are semantically annotated with RDF meta-data. To enable machines to access the functionality, Semantic Web Services are used. Semantic Web Services enable the formal specification of services, allowing their automated, goal-driven discovery and usage [5].

In this paper we sketch out an architecture of a *Semantic Portal* that provides a common front-end to the information and functionality of individual Semantic Web applications. As scenario to illustrate the Semantic Portal in this paper we take the domain of cultural events. In the subsequent sections we introduce step by step the parts of the architecture and highlight the added value gained by the use of Semantic Web technologies. Parts of this architecture are already implemented in the WEESA project, other parts are still future work.

In Section 2 we start with the semantic annotation of Web applications and in Section 3 we propose to provide these annotations in a knowledge base (KB) for download and querying. In Section 4 we introduce the Semantic Harvester which is used to dowload the KBs from the individual Semantic Web applications in the domain and to integrate the collected meta-data in the global KB of the Semantic Portal. In Section 5 we use Semantic Web Services to enable the Semantic Portal to access the functionality of the Web applications and Section 6 concludes the paper.

2 Semantic Annotation of Web Pages

As a first step towards Semantic Web applications, the Web pages have to be semantically annotated with RDF meta-data. This enables machines to have access to the content of the pages. Several tools such as the SHOE Knowledge Annotator [9], the CREAM OntoMat [8], and SMORE [10] have been proposed to support the user when annotating existing Web pages. To manually annotate Web pages is only feasible if the number of Web pages is small and the content does not change frequently.

To annotate dynamic Web pages, that obtain their content from a background logic such as a database, the annotation process should be integrated in the engineering process of the Web application. During the engineering of a Web application, information items can be identified more easily than in the generated HTML Web pages. Once the information item is identified, it is mapped to a concept defined in the ontology. For example, the result of a database query is mapped to a property in the ontology, to indicate that a new RDF statement has to be generated with the query result as value for the property.

In the WEESA project (WEb Engineering for Semantic web Applications) [18, 19] we developed a technique to integrate the semantic annotation task of an *XML based Web application* into the engineering process. XML based Web applications use XML documents for the content and XSLT stylesheets to transform the XML document into the HTML Web page. In WEESA we do not annotate the HTML page, since it is hard to identify information items in the HTML code that mainly describes the graphical appearance of the Web page. Instead, we annotate the XML document whose semi-structured nature makes it easier to identify information items. In WEESA the structure of the XML document is used to identify XML elements/attributes which are mapped to concepts in an ontology. The mapping definition is a manual process that has be done when developing the Web application. The mapping is then taken to automatically generate RDF meta-data from XML content documents.

At the design level of the Web application only the structure of the XML document is known and no XML document instances are available. Therefore we use the structure information from the XML Schema to define the mapping to the ontology. Figure 1 shows the definition of the WEESA mapping at the design level of the Web application and how this mapping is used at instance level to automatically generate RDF meta-data and the HTML page from XML

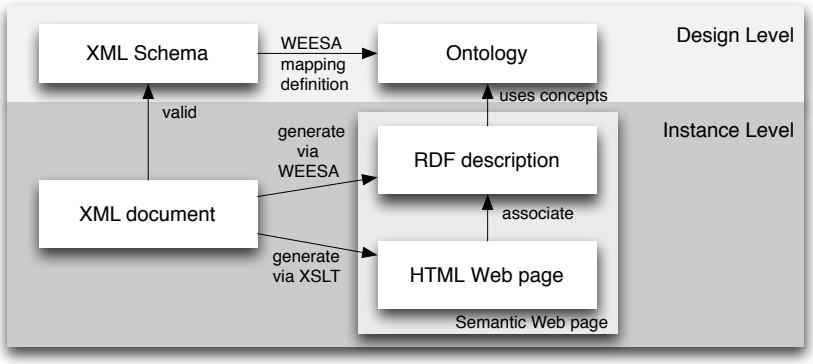

Fig. 1. Definition of the WEESA mapping at the design level and RDF meta-data generation at the instance level.

documents. Finally, the HTML page and its RDF meta-data description have to be associated (e.g. the HTML page links to its RDF description) [16]. To allow developers to easily take advantage of WEESA, we integrated the WEESA meta-data generator in the Apache Cocoon Web development framework [4]. A detailed description of WEESA can be found in [18, 19].

Once the Web pages are semantically annotated, the content can be further processed by other applications without loosing its semantics. For example, we developed a Semantic Clipboard [15] that can be used to copy and paste the RDF annotations of Web pages to desktop applications. In contrast to the clipboards of current operating systems, the Semantic Clipboard preserves the meaning of the data. For example, a Web application for cultural events offers an online ticket store. When a user buys a ticket and wants to enter the event into his calendar, he has to manually create the new calendar entry. With the Semantic Clipboard the user can directly copy and paste the ticket receipt Web page to the calendar and the entry is generated automatically.

3 Meta-Data Model of the Web Application

Looking at the meta-data of a single Web page, however, gives only a limited view on the available meta-data. For querying and reasoning purposes it is better to have the meta-data model of the whole Web application as *knowledge base* (KB) at hand. The same strategy is followed in search engines on the Web. A query is answered with the help of an indexed cache and not by starting to extract information from pages on the Web.

Ontobroker [7], SEAL [11], and SHOE [9] use a KB to enable reasoning and database-like queries. CREAMS's OntoMat [8] uses the knowledge base to assist the semantic annotation from documents. All the systems mentioned above use a

Web crawler to build their KB. The Web crawler periodically collects information from semantically annotated Web pages and adds the meta-data to the KB.

Using a crawler enables that the Web application and the application using the KB are loosely coupled and can be distributed over the Internet. This loose coupling, however, makes it difficult to track the deletion or update of a Web page. The KB can therefore become inconsistent with the actual information available on the Web server.

To overcome this inconsistency problem we suggest to generate the KB on the server side and to offer the KB as a Web Service for querying. In the WEESA project we extended the WEESA meta-data generator to not only generate the RDF graph, but write the generated meta-data to the KB [18]. The accumulated meta-data in the KB is then offered as service for SPARQL queries [17]. In addition a snapshot of the KB is be offered in compressed format for download by clients. In this case the KB of a Web application can be downloaded as a single stream rather than requiring separate requests for every single Web page. This follows the idea proposed by the Harvester search engine architecture [1] where the index of Web applications is created locally at server side and is then offered to brokers for download. The download of the KB, however, leads again to the inconsistency problem discussed above. But still, if the downloaded KB and the query service to the KB offered by the Semantic Web application are used in symbiosis, the application can benefit from it. Static information that does not change frequently can be retrieved from the local copy of the KB without online access to the Web application. Dynamic information that changes frequently can be retrieved from the query service on demand.

This loose coupling between the Web application and the KB, however, leads to problems when maintaining the meta-data model and keeping it up-to-date. In the following we discuss these problems and our solution in detail. Figure 2 shows the components that are responsible to keep the KB up-to-date.

Populating the KB: With the proposed procedure the KB is incrementally filled. Each time a Web page is requested its meta-data is written to the KB. This means, we initially start with an empty KB. To generate the KB of the whole Web application, each available Web page has to be requested. We use the Cocoon command line interface for this purpose. A built-in Cocoon crawler follows every internal link, requests the Web page, and therefore causes the meta-data generation. This is used to initially fill the KB.

Page Deletion Handling: When a Web page has been deleted, the corresponding RDF statements have to be removed from the KB. Therefore, the KB management has to be informed, when a requested Web page cannot be found. In this case the Web server sends a 404 "not found" page back to the client. In the Cocoon configuration we can catch the "not found" event and inform the KB management about the URL of the page that was not found. The KB management can then cause the deletion of all RDF statements that originated from this Web page. To do so, the KB management has to be able to identify the RDF statements that were generated for a given URL. This

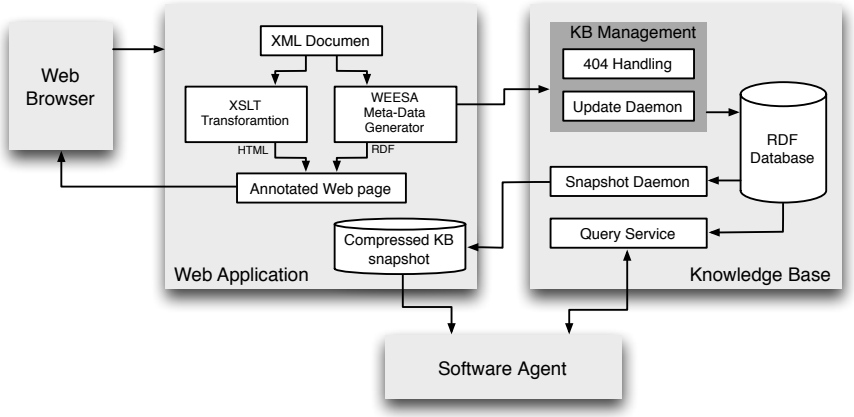

Fig. 2. Accumulating the meta-data of all Web pages in the KB of the Web application.

means we have to store the origin information of each RDF statement in the KB. This is done using RDF named graphs [3].

Content Change Handling: When an editor changes the content of a Web page, he typically browses to the page and check for the correct presentation of the changes. In this case, the RDF meta-data is generated and an update in the KB is issued. The update first removes all RDF statements from the KB that originated from the Web page with the given URL and adds the newly generated meta-data.

However, if the content of the background database of the Web application is changed and the dynamic Web page that is based on the database is not queried, no update to the KB is issued and the KB becomes inconsistent with the content of the Web application. To avoid this, we add the last modification time and the update interval to the RDF statements for each Web page. This enables the Update Daemon in the KB management to periodically check for outdated RDF statements and to issue an update with the data from the Web application.

An attribute in the WEESA mapping definition is used to define the update interval. The update interval informs the Update Daemon about the frequency the RDF statements that originated from this mapping definitions have to be updated. For static pages the update attribute can be omitted. In this case, the update daemon does not check for updates for this kind of Web page.

Filtered Meta-model building: It is not necessary to add the meta-model of each Web page to the KB of the Web application. For example, the meta-data of the personal shopping cart page should not be added. But adding meta-data to such a page is still useful. Therefore an attribute is added to the WEESA mapping definition to instruct the WEESA meta-data generator not to write the meta-data to the KB for these types of pages.

Providing access to up-to-date information: When a snapshot of the KB is downloaded by a client and queried locally, the local copy of the KB and the actual data of the Web application may become inconsistent. For some types of information this is not a problem since it is rather static and does not change often. The composer and the name of an opera are examples of information that does not change often. However, other types of information such as the number of available seats of a performance might change frequently and it may be necessary to have access to latest data from the Web application. Therefore we use RDF named graphs to store the URL to the RDF graph of the Web page. This way the client is able to request the actual RDF graph on demand.

To keep the Semantic Web application scalable, the Cocoon caching mechanism is used. When the XML document, the Web page is based on, has not been changed since the last request, the subsequent requests are served by the Cocoon cache. This way the WEESA meta-data generator is not processed and no update is issued to the KB.

4 Using the Semantic Harvester to build the KB for a Semantic Portal

In the previous section we argued that providing the meta-data descriptions of all Web pages in a KB enables database-like queries against the Semantic Web application. We further provide software agents the possibility to download a snapshot of the KB for local processing. In this section we introduce the *Semantic Harvester* that is used to build the global KB of a *Semantic Portal*.

The goal of the Semantic Portal is to offer a unique front-end to individual Web applications in a specific domain, for example cultural events. The portal integrates information from several Web applications and presents this information in a unique style on its Web pages. The portal further enables the user to issue database-like queries not only to a single Web application but to all Web applications that contribute to the portal.

To contribute to the Semantic Portal, Semantic Web applications have to fulfill two requirements. First, they have to be from the same domain and use the same ontologies to annotate their Web pages or ontologies that can be mapped to the ontologies used by the portal [6]. Second, the Web application has to offer its meta-data in a KB for querying and download, as introduced in the previous section.

To build the KB of the Semantic Portal we use the Semantic Harvester. The Semantic Harvester is based on the idea of the Harvester search engine architecture [1]. Semantic Web applications that want to contribute to the Semantic Portal register at the Semantic Harvester. The Harvester then downloads periodically the compressed snapshot of the KB of each registered Web application and integrates the meta-data into the global KB of the Semantic Portal. If a Web application uses an ontology for their semantic annotation that is different to the

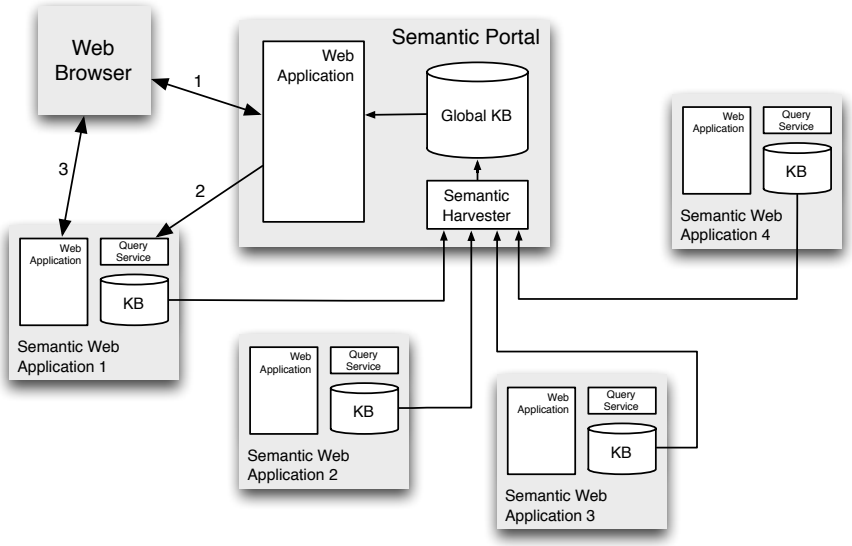

Fig. 3. Architecture of of the Semantic Portal infrastructure.

one used by the portal, ontology mediation is used to integrate the meta-data into the global KB [2, 13]. Figure 3 shows the Semantic Harvester that generates the global KB of the Semantic Portal.

The Web front-end of the Semantic Portal uses the global KB, that was created by the Semantic Harvester, to generate its Web pages in a unique look and feel. The user does not recognize that the presented information originates from different sources. The user interaction with the portal is indicated by arrow 1 in Figure 3. This way the portal can provide database-like queries over distributed individual Web applications. For example, our cultural portal is able to offer a search interface for an event in a given city on a given date over all involved Web application.

Depending on the domain, some information is rather static whereas other information changes frequently. Since the programmer of the Semantic Portal knows the domain and the ontologies that are used to model the domain, he knows which information is static or dynamic. Static information can be directly taken from the global KB. Dynamic information, for example if tickets are available for a given event, changes frequently and is therefore retrieved directly from the Semantic Web application via the Query Service (shown by arrow 2 in Figure 3). When a user in the next step wants to know the source of the information, the portal uses the origin information in the KB that was added using RDF named graphs, to redirect the user to the Web application that provides the according Web page. This interaction is shown by arrow 3 in Figure 3. The user now can, for example, buy tickets for the selected event.

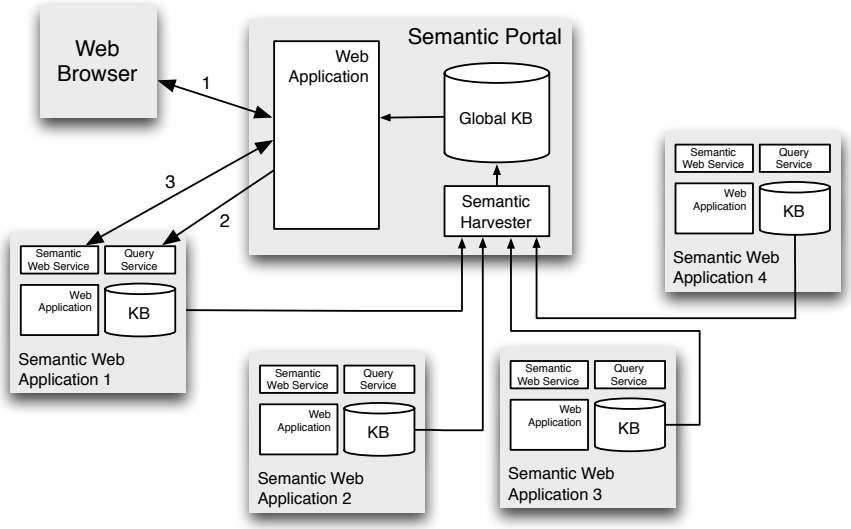

Fig. 4. Architecture of the Semantic Portal using Semantic Web Services.

5 Semantic Web Service

In the Semantic Web application we sketched out so far, only the information is accessible for machines. The services the Web application offers, for example the online ticket store, can still only be used by human users. The Semantic Portal, introduced in the previous section, does not have access to the service. To make the service machine accessible the service has to be described as a Semantic Web Service. The semantic markup of Web Services provides a declarative, computer-processable API for selecting and executing a service [12].

To make the services of the Web applications accessible by the Semantic Portal, we propose to add the semantic service description to the compressed snapshot of the KB. This way the Semantic Harvester downloads the service description of all contributing Semantic Web applications and the Semantic portal has access to the services descriptions without consulting a service discovery service first.

Several languages have been proposed to the W3C to specify a Semantic Web Service: the OWL Web Ontology Language for Services (OWL-S) [14], the Web Service Modeling Ontology (WSMO) [22], the Semantic Web Services Framework (SWSF) [20], and the Web Service Semantics - WSDL-S [21]. Since we have not yet started to implement the Semantic Web Service part we have no yet decided which language we are going to use.

Once the functionality of the Web application is described with semantic markup as a Semantic Web Service, the Semantic Portal can directly offer the services on the portal sites without redirecting the task to the Web application.

The Semantic Portal can therefore offer a unique interface to the functionality of the contributing Semantic Web application. This is shown with arrow 3 in Figure 4. Taking our cultural portal example, the portal can offer a common user interface to the online ticket shops of all involved Semantic Web applications. It becomes transparent for the user which Web application he is using.

6 Conclusion

In this paper we introduced the architecture for a Semantic Portal which provides a common interface to the content and services of the Web applications in a given domain. We discussed step by step the requirements to the Web applications to offer their functionality via the portal site. The Web applications have to provide semantically annotated Web pages that use the same ontologies for their annotations or ontologies that can be mapped to the ontologies used by the portal and and offer their annotations in a KB for querying and download. In addition, the functionality of the Web application is described with Semantic markup as a Semantic Web Service. This way the Semantic Portal can provide database-like queries and a common interface to the services of individual Semantic Web applications.

References

1. C. M. Bowman, P. B. Danzig, D. R. Hardy, U. Manber, and M. F. Schwartz. The harvest information discovery and access system. In *2nd International World Wide Web Conference*, Chicago, Illinois, USA, October 1994.
2. A. E. Campbell and S. C. Shapiro. Ontolic mediation: An overview. In *Proceedings of the IJCAI95 Workshop on the Basic Ontological Issues in Knowledge Sharing*, Montreal, Canada, 1995.
3. J. J. Carroll, C. Bizer, P. Hayes, and P. Stickler. Named graphs, provenance and trust. In *Proceedings of the 14th International Conference on World Wide Web*, pages 613–622, Chiba, Japan, 2005. ACM Press.
4. The Apache Cocoon project homepage, Last visited February 2005. `http://cocoon.apache.org/`.
5. J. de Bruijn, D. Fensel, U. Keller, and R. Lara. Using the web service modeling ontology to enable semantic e-business. *Communications of the ACM*, 48(12):43–47, 2005.
6. M. Ehrig, J. Euzenat, E. Franconi, P. Hitzler, M. Krötzsch, L. Serafini, G. Stamou, Y. Sure, and S. Tessaris. D2.2.1 v2 specification of a common framework for characterizing alignment. Technical report, Knowledge Web, 2005.
7. D. Fensel, J. Angele, S. Decker, M. Erdmann, H.-P. Schnurr, S. Staab, R. Studer, and A. Witt. On2broker: Semantic-based access to information sources at the www. In *Workshop on Intelligent Information Integration at the International Joint Conference on Artificial Intelligence*, Stockholm, Schweden, August 1999.
8. S. Handschuh and S. Staab. Annotation of the shallow and the deep web. In S. Handschuh and S. Staab, editors, *Annotation for the Semantic Web*, volume 96 of *Frontiers in Artificial Intelligence and Applications*, pages 25–45. IOS Press, Amsterdam, 2003.

9. J. Heflin, J. Hendler, and S. Luke. Shoe: A blueprint for the semantic web. In D. Fensel, J. Hendler, H. Liebermann, and W. Wahlster, editors, *Spinning the Semantic Web*, pages 29–63. The MIT Press, 2003.
10. A. Kalyanpur, J. Hendler, B. Parsia, and J. Golbeck. SMORE - semantic markup, ontology, and RDF editor. Technical report, University of Maryland, 2003. `http://www.mindswap.org/papers/SMORE.pdf`.
11. A. Maedche, S. Staab, N. Stojanovic, R. Studer, and Y. Sure. SEmantic portAL: The SEAL approach. In D. Fensel, J. Hendler, H. Liebermann, and W. Wahlster, editors, *Spinning the Semantic Web*, pages 317–359. The MIT Press, 2003.
12. S. A. McIlraith, T. C. Son, and H. Zeng. Semantic web services. *IEEE Intelligent Systems*, pages 46–53, March/April 2001.
13. N. F. Noy and M. A. Musen. Evaluating ontology-mapping tools: Requirements and experience. In *Proceedings of OntoWeb-SIG3 Workshop at the 13th International Conference on Knowledge Engineering and Knowledge Management*, Siguenza, Spain, 2002.
14. OWL web ontology language for services (OWL-S). Submission request to W3C, Last visited March 2006. `http://www.w3.org/Submission/2004/07/`.
15. R. Paizoni. Semantic clipboard. Master's thesis, Vienna University of Technology, 2005.
16. S. B. Palmer. RDF in HTML: Approaches, June 2002. `http://infomesh.net/2002/rdfinhtml/index.html`.
17. E. Prud'hommeaux and A. S. eds. SPARQL query language for RDF. W3C Working Draft, 19 April 2005. `http://www.w3.org/TR/rdf-sparql-query/`.
18. G. Reif. *WEESA - Web Engineering for Semantic Web Applications*. PhD thesis, TU Vienna, 2005. `http://seal.ifi.unizh.ch/fileadmin/User_Filemount/Publications/reif-phdthesis05.pdf`.
19. G. Reif, H. Gall, and M. Jazayeri. WEESA - Web Engineering for Semanitc Web Applications. In *Proceedings of the 14th International World Wide Web Conference*, pages 722–729, Chiba, Japan, May 2005.
20. Semantic web services framework (SWSF). Submission request to W3C, Last visited March 2006. `http://www.w3.org/Submission/2005/07/`.
21. Web service semantics - WSDL-S. Submission request to W3C, Last visited March 2006. `http://www.w3.org/Submission/2005/10/`.
22. Web service modeling ontology (WSMO). Submission request to W3C, Last visited March 2006. `http://www.w3.org/Submission/2005/06/`.