Scalable Matchmaking for a Semantic Web Service based Architecture

Nicola Henze and Daniel Krause
Distributed Systems Institute, Semantic Web Group, University of Hannover
Appelstraße 4, 30167 Hannover, Germany
{henze,krause}@kbs.uni-hannover.de

Abstract

We propose a two-step matchmaking procedure for a Web Service oriented architecture to cope with scalability problems if a large amount of Web Services has to be processed. We distinguish between domain-aware and domain-independent matchmaking, and show how such a two-step matchmaking process can be realized. We discuss the approach within the Personal Reader architecture which enables the use of Web Service based applications to personalize Semantic Web content.

1. Introduction

While more and more data became machine readable over the last years, the Semantic Web Stack [1] does not include any kind of application layer that uses these data to base applications on them.
At this point of time the usage of Web Services is the most promising approach to fill up this gap in the Semantic Web Stack. The main advantages of Web Services are their platform and location independence. Web Services are accessed via standardized Hypertext Transfer Protocol and therefor can be stored and used on every web server in the world. Web Services can build up their functionality utilizing Semantic Web content or other Web Services. By combining and syndicating different Web Services which offer basic functionalities, the realization of more complicate processes is possible. As every Web Service can physically be located on any web server in the world, the definition of the interfaces that need to be implemented require most attention.

Furthermore, as Web Services encapsulate functionalities, the reuse of functionalities – encapsulated in the Web Services – becomes feasible. The reuse of functionality by accessing Web Services is based on the assumption that Web Services are long-term available. But experiences from the WWW show that a network where many different people can interact in by creating own Web Services, is a quick changing environment. In the Semantic Web not only content changes quickly, but also Web Services that process the content might appear and disappear in a frequent manner.
Combining Web Services in such a dynamic environment requires that the combination process itself has to be dynamic. This means that Web Services have to discover other Web Services automatically, and need to be able to detect those Web Services that offer the momentarily required functionality. To support such a dynamic combination, the functionality of the Web Service has to be described *semantically* in a way that enables an automatic matchmaking between the requested and the offered functionalities.
In this paper we present the Personal Reader architecture that offers different kinds of Web Services: Syndication Services provide application features (like user interfaces and data syndication), Personalization Services provide personalization functionality (like e.g. the provision of contextual information, recommendations, etc.) in the Semantic Web. For describing the semantics of invocation and response parameters we introduce *configurable descriptions* which enable a two-step matchmaking to discover appropriate Web Services with a high efficiency and scalability.

2. The Personal Reader Architecture

Our approach for a Semantic Web architecture offers users a uniform entry point to access the Semantic Web, and in particular to Web services in the Semantic Web. It has been realized as part of the *Personal Reader Framework* [2, 3] which offers an environment for designing, implementing and realizing Web content readers in a service-oriented manner (see Figure 1):

- Web services deliver personalized recommendations for content, extracted and obtained from the Semantic Web. Using Semantic Web techniques they describe their offered functionality in a machine processable format. Optionally, they can return visualization templates that can be used to create user interface snippets for presenting the results of the Web services. We call these kind of Web services *Personalization Services*,

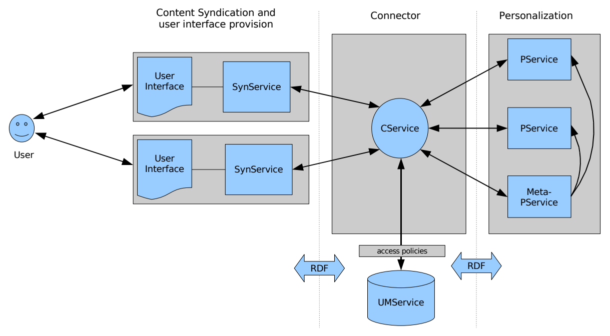

Figure 1. Overview of Personal Reader architecture

PServices for short.

- *Meta PServices* can be employed to have single entry points to cooperating or concurrent PServices. Meta PServices offer the technical platform in the Personal Reader architecture to orchestrate simple and generic PServices to model even complicated domains or relationships.

- For syndicating the results of PServices and for creating appropriate user interfaces, *Syndication Services* (*SynServices* for short), are responsible for displaying the results of several PServices and/or Meta PServices to the user. Each SynService provides at least one user end point (that is also called user interface) to a certain domain or task, and allows the user to benefit from many PServices simultaneously, which are selected, combined and customized as the user wishes. SynServices can be realized as RDF browsers, can implement their own RDF processing interface, or they can make use of visualization templates provided by the different PServices / Meta PServices.

- User interfaces are responsible for the interaction with the user. They receive XML or RDF messages from the SynService and visualize them according to the display device. Every user interface deals with one special class of devices, for example PCs, PDAs or mobile phones which means that device adaption is dealt with by the user interfaces. Therefore, one SynService can have many different user interfaces.
The user can interact with the user interface by generating events, like clicking on a button, entering text in a form etc. These events are sent back to the SynService that processes these events.

- For enabling the whole process, a core information provider – a user modeling service, $UMService$ for short, deriving appropriate user profiles – is essential.
In our architecture, the UMService realizes a centralized approach for user modeling, maintaining and protecting information about a user on behalf of this user. The main reason for choosing a centralized approach is to realize privacy protection: the user has full control about his user profile, and can define policies on which parts of this user profile might be public, or are available to trusted parties only. One central instance for all this information makes it possible to create awareness about stored information, and on realizing policy-protected access. Furthermore, this central UMService receives updates from SynServices or PServices in different domains, and allows for cross-domain re-use of user profile information (if the user wants that).

- From a technical point of view, another component is required to maintain the communication between the SynServices providing the user interface, the PServices, and the UMService. This is the so-called *Connector Service* (*CService* for short) which harvest Web service brokers, collects information about detected PServices (for discovery, selection, customization, and invocation), and for organizing the communication between all involved parties, including requests to the UMService.

2.1. Usage Scenario

To describe the relationship between the different kind of web services we point out which steps need to be performed until the user gets his personalized content visualized in the user interface:
First, the user logs in at the UMService, using his username and password. Then UMService creates a session ID (SID for short) which will be valid during the whole session, and can only be traced back to the user at the UMService (realized via public-private-key encryptions). After logging in, the user accesses an entry point. The SynService , that belongs to the entry point, calls the CService and receives a list of available SynServices, a human readable description and the URL of the user interface that is appropriate to the user's display device. Entry points can give different levels of support to make the selection of the SynServices more easy for the user. Possible entry points are:

- The entry point can be an agent and ask the user what task he wants to fulfil and to recommend SynServices that cope with this issues.

- Another entry point can build a hierarchical organized portal, grouping similar SynServices.

- A third entry point may just returns an unordered list of all available SynService.

The user selects the SynService that fits his tasks best and is redirected to the URL of the user interface that belongs to the selected SynService which is able to visualize the content according to the user's display device. Furthermore, by accessing the user interface, the SID is passed to it and along to the SynService. The SynService requests a list of available PServices from the CService. In this request the SynService also submits which Ontologies it is able to handle. The CService does the first step of the matchmaking and returns PService candidates. In the second step of the matchmaking, the SynService selects PServices according to their description of offered functionality.
To invoke the selected PServices the SynService requires invocation parameters to personalize the results of the different PServices according to the user's preferences. These invocation parameters can be gained in two different ways:

- The SynService asks the UMService if it has stored information how the user has specified a value for a parameter beforehand. It is used in the confidence that this previous value will be valid for the current invocation. If the confidence is too low or there are no values stored until now, the user is asked directly.
- The SynService asks the user directly via the user interface to define a value for the required invocation parameter.

If both ways fail, for example if no information is stored in the UMService and the user rejects to enter a value, and the invocation parameter is marked as "required" in the description of the PService, this PService is omitted by the SynService. Afterwards, the SynService invokes the remaining PServices by sending an invocation request to the CService. The CService invokes every PService synchronous. The PServices use the invocation parameters and additionally, can send requests to the UMService to get more user specific data. Afterwards, they return the results to the CService that combines all single responses to one response and sends it back to the SynService. The SynService combines, filters and enriches the response and formats it in a way that user interfaces can easily visualize the response. This response is sent from the SynService to the user interface and is displayed to the user.
In an iterative manner the user or the SynService now can alter and refine invocation parameters to receive different or better results.

2.2. User Modeling

As illustrated in the previous chapter two types of Web Services interact with the user: The SynService as it gets input from the user via events and the PService that gets personalized invocation parameter. This enables both types of Web Services to derive properties about the user with the help of different user modeling techniques. As both types of Web Services operate in one special domain they both should have detailed knowledge of the domain that they can take into account when modeling the user.

2.3. Semantic Web Services

To enable automatic matchmaking we require a machine-readable description of the functionality of our Web Services. Our approach relays on a semantic abstraction of a datatype-based description: instead of using datatypes like strings, integer, etc. to describe single parameters we group these parameters into semantic parameters that are described by Ontologies in a Configurable Description: For example, some parameters `keyword1, keyword2,..., keywordn` build the semantic parameter `query` that is stored in the Configurable Description. By defining the ontology, that is used for the Configurable Description, precise enough, invocation and response parameters are sufficient to describe the functionality of the whole Web Service. Furthermore, as the Configurable Description contains the vocabulary that the Web Service is able to process, it is the groundwork for our two-step matchmaking.

3. Matchmaking

In a Web Service oriented architecture matchmaking between descriptions of Web Services and requirements is always domain aware. That means, that programs that do the matchmaking require domain knowledge. Thus, there are two possible realizations for matchmaking:

- *Centralized matchmaking*: One programs knows all domains that are used in the architecture.
- *Decentralized matchmaking*: For every single domain, and for all domains which are commonly used together, there has to be an own matchmaking program.

The first solution would run into problems if new Ontologies appear, because every new Ontology determines an update of the matchmaking system. Furthermore, this approach does not scale as more and more Ontologies appear, the program will result in a more and more complicated system which makes it unmaintainable.
The second solution scales very well in terms of the appearance of new Ontologies as existing systems have not to be changed but only new ones are added. The problem to cope with in this solution is the number of Web Services: The larger the number of available Web Services is, the longer the matchmaking process takes.
Our solution for this problem is to introduce a two-step

	Centralized matchmaking	Decentralized matchmaking	Two-step matchmaking
adding new domains	difficult	good	good
handling large amounts of Web Services	difficult	difficult	good

Table 1. Comparison of different matchmaking approaches

matchmaking: In the first step our central Web Service, the CService, does a domain-independent matchmaking by checking whether two Web Services can handle the same Ontology. This is done by analyzing the Configurable Description of the Semantic Web Services. All Web Services that are able to handle the required Ontology are marked as candidates and passed to the second step of the matchmaking process. This first step is very efficient as it is not more than a simple RDF database lookup in our Configurables database.
In the second step of matchmaking, we use a domain-aware matchmaking program like in the second realization. This is done by the SynServices, that know best which functionalities, expressed by the description of input and output parameters in the Configurable Description, are required.
This solutions scales in two ways: On the one hand the appearance of new Ontologies does not require changes on the centralized component. On the other hand, the first step, that scales very well in term of handling large amounts of Web Services, slashes the amount of Web Service that are passed to the second step. Therefore, the time that is required for the matchmaking process is little influenced by adding new domains to the system.

4. Discussion of the approach and related Work

The approach presented in this paper differs from common usage of Web Services, and shows how Web Services can be used as first class citizens in a Semantic Web. In our thinking, Web Services provide functionality snippets, which shall be plugged together to support a user during the task he is currently performing. Thus, Web Services provide functionality in the Web for *end users*, and our approach shows how user interfaces for accessing, plugging together, syndicating and using Web Services can be realized. The different stakeholder of the process are the Web Services which take care on appropriate user interfaces (Syndication Services), and those who provide the functionality snippets (the Personalization Services). For performance and reusability issues, a communication facilitator has been employed (the Connector Service).

Related work to our approach can be found in projects which apply Web services in the Semantic Web: Current research here focuses more on enabling technologies like Web services discovery, composition and orchestration (cf. [4, 5]). However, approaches which focus on usability and user-interfaces for accessing a Web Service-oriented Semantic Web are today missing.

5. Conclusion and Further Work

In this paper, we have proposed an approach to realize personalized access to Web Services in the Semantic Web. We have identified the main challenges to overcome, and especially discussed the influence of domain dependent vs. domain independent discovery, selection and invocation of Web Services. We are currently extending the architecture of the Personal Reader Framework as discussed in the paper to realize the full functionality of the configurable descriptions and the matchmaking processes. A prototype showing the applicability of the proposed solutions has already been realized and demonstrates the syndication of various podcast providers according to the music interests of users[1].

References

[1] Berners-Lee, T. Semantic Web - Keynote at XML 2000 Conference, 2000. http://www.w3.org/2000/Talks/1206-xml2k-tbl/slide10-0.html.

[2] Henze, N., and Krause, D. Personalized access to web services in the semantic web. In *SWUI 2006 - 3rd International Semantic Web User Interaction Workshop, colocated with the 5th International Semantic Web Conference* (nov 2006).

[3] Henze, N., and Kriesell, M. Personalization Functionality for the Semantic Web: Architectural Outline and First Sample Implementation. In *1st International Workshop on Engineering the Adaptive Web (EAW 2004)* (Eindhoven, The Netherlands, 2004).

[4] McIlraith, S., Son, T., and Zeng, H. Semantic web services. *Intelligent Systems 16*, 2 (2001), 46–52.

[5] Motta, E., Domingue, J., and Cabral, L. Irs-ii: A framework and infrastructure for semantic web services. In *Proceedings of the 2 Intl. Semantic Web Conference* (Florida, USA, 2003).

[1] The prototype can be accessed via www.personal-reader.de/agent/