Web Services for Transmitting Product Information in the Context of Business-to-Business Integration

Stefan Kuhlins and Bjørn-Henrik Zink

University of Mannheim, Chair of Information Systems III, D-68131 Mannheim, Germany
{kuhlins, bhzink}@rumms.uni-mannheim.de

Abstract. Imagine that the product data of every online shop would be machine-readable and you could find the best product offers in an instant. Many e-business scenarios, such as price comparers, electronic markets, product search engines, and shop directories require integration of distributed heterogeneous information systems. Information extraction from heterogeneous Web sites has been a research topic for a long time. A frequently used approach for automatic information aggregation is to parse and extract information directly from Web pages. The main problem of these software routines is to find appropriate methods for parsing HTML pages. Web services on the other hand enable software routines to directly communicate with distributed information systems, therefore there is no need for parsing Web pages anymore. This paper examines new opportunities and challenges related to real-time product information transmission using Web services. The minimal characteristics that are common to all Web services and their application to a collaborative scenario between price comparers and online shops will be presented. In order to automate the information system integration we suggest that it is necessary to standardize the Web service interface for accessing product information from online shops. This is demonstrated by an example implementation. The main contribution of this paper is the practical aspect of using Web service semantics and technologies for real-time product information transmission.

1. Introduction

Various collaborative business scenarios, such as electronic markets, product search engines, and shop directories rely on an exchange of information between distributed heterogeneous information systems. Web services enable data aggregation and integration of heterogeneous information systems over the Internet through open standards that are widely supported [HMS02, Ga02]. Currently, there is little direction available on how to best apply Web services to collaborative business scenarios with multiple service providers and service requesters. Furthermore, it is interesting to investigate whether or not Web services are applicable to small companies like online shops with moderate IT resources and budgets. In this paper, we focus on the scenario of price comparers and online shops. Both seek to find an inexpensive and fast way of integrating their information systems with each other. Automation of the integration process would contribute significantly to pursue this goal.

Automatic information extraction from heterogeneous Web sites has been a research topic for a long time [HMS02]. A frequently used approach for automatic information aggregation is to parse and extract information directly from Web pages, often referred to as wrappers [Ei99, KT92]. The emphasis of these software routines is to develop suitable methods for parsing HTML pages. Unfortunately, when HTML pages change the wrapper must be adjusted manually to avoid malfunction [MSZ01]. Due to the fact that HTML does not separate Web content from Web presentation HTML pages change rather frequently and are complicated to interpret automatically.

XML based Web services represent another way of collecting information from heterogeneous information systems. XML separates content from presentation but lacks semantics [MSZ01]. The *World Wide Web Consortium* (W3C) has added semantics to XML through developments such as *XML Schema*, *Resource Description Framework* (RDF), *RDF Schema*, and *OWL Web Ontology Language* (OWL).[1] Furthermore, XML based initiatives, such as *Simple Object Access Protocol* (SOAP), *Web Service Definition Language* (WSDL), *Universal, Description, Discovery, and Integration* (UDDI) and *electronic business XML* (ebXML) *Registry*, improve automatic Web services discovery and execution [MSZ01].[2] Despite these advances, it cannot be certain that software routines can determine the intended interpretation of Web services operations. For example, it would be cumbersome for software routines to identify that a WSDL operation `getSalery` refers to the same information as a WSDL operation `getWage`.

It would be possible to solve this problem by defining an ontology for product information interfaces. Our vision, however, is a multitude of standard interface descriptions that other business applications can reuse. Besides solving the above described execution problem it would support automatic implementation, deployment, and discovery. In this paper, we will present a simple WSDL interface for transmitting product information as a typical example.

The remainder of the paper is structured as follows: Section 2 presents the business scenario. In Section 3 the common characteristics of Web services are briefly described. Furthermore, we explain the process of creating Web services. The following parts of Section 3 elaborate on the steps of the Web service creation process. Finally, Section 4 summarizes and draws conclusions on the use of Web services for the business scenario in question.

2. Scenario

Elm@r, the electronic market, is a reference implementation for the `shopinfo.xml` standard and is part of a research project at the University of Mannheim's chair of *Information Systems III*. There are currently more than 190 participating shops and

---

[1] http://www.w3.org/, http://www.w3.org/XML/Schema, http://www.w3.org/TR/rdf-concepts/, http://www.w3.org/TR/rdf-schema/, http://www.w3.org/TR/owl-features/

[2] http://www.w3.org/TR/soap12, http://www.w3.org/TR/wsdl, http://www.uddi.org/, http://www.ebxml.org/

650,000 products in the *Elm@r* database[3]. *Elm@r* offers a product and vendor search as well as a price comparison mechanism that enables users to get an overview of product offerings. *Elm@r* presents price comparison along with information on availability, delivery time, delivery costs, and so on. The price comparison mechanism works through a combination of database search and real-time requests. When an online shop registers with its shopinfo.xml at *Elm@r*, it has two options for sharing its product information. First, the online shop can make a product file available for download. Second, the online shop can enable real-time product requests based on well-known techniques used by HTML forms, but create XML documents as response. The real-time product requests are dynamically executed at the time a user starts searching. It should be emphasized that the focus of this paper is the transformation of this simple real-time request into Web services.

Price comparers aim at providing consumers with an overview of price and availability of products. In order to pursue this goal price comparers need access to information from online shops. There are several issues related to the integration of online shops. First, a price comparer has to discover online shops and keep track of them in the future. Second, having found online shops the price comparer has to integrate its information system with online shops. Assuming that online shops run their applications on heterogeneous systems that are implemented in various programming languages, integration becomes very cumbersome. Thus, it would be desirable to find a solution that automatically discovers online shops and integrates with them.

Online shops want to reach as many potential customers as possible because this raises the chances of selling products. One way of doing so is by listing product information at price comparers. The question is how to make price comparers aware of the online shop in an inexpensive and fast manner. Furthermore, the integration issue mentioned in the previous paragraph can also be applied to online shops, however, the opposite way around. With the relatively modest IT budget of most small online shops, it cannot be expected that they will provide all price comparers with a specific interface. In other words, online shops must find an inexpensive way to make price comparers aware of them and come across a technical solution that does not require highly sophisticated technical skills. Automation would be eligible for the publishing and integration process.

The difference of this situation to other Web services scenarios can best be described through an example. For instance, various service providers can implement their version of a stock quote Web service. Despite the different interface definitions, a service requester can expect to get the same result from any of these stock quote Web services. Thus, the service requester can choose to integrate with any of the stock quote Web services without loss of information. In the case of price comparers and online shops, however, the service requester obtains unique information from each Web service. Price comparers would have to develop a Web service client for all online shops in order to avoid loss of information. With thousands of online shops, this is obviously very cumbersome and error-prone. Whereas standardization is not necessary in cases of Web

---

[3] http://www.elektronischer-markt.de/

services similar to the stock quote example, it surely seems worthwhile to standardize the interface for accessing product information from online shops.

3. Web Services

For our purpose, we define Web service in accordance with [WSA04] as a software system designed to support interoperable machine-to-machine interaction over a network. It has a machine-processable WSDL interface. Other systems interact with the Web service in a manner prescribed by its description using SOAP messages, conveyed using *Hypertext Transfer Protocol* (HTTP) with an XML serialization in conjunction with other Web-related standards.

The common process of engaging a Web service contains four steps [WSA04]:

1. Define semantics and service description.
2. Service requesters and service providers must become known to each other. For instance, this can be done by registering and querying an UDDI registry.
3. The service description and semantics are input to, or embodied in, both service requesters and service providers as appropriate.
4. Service requesters and service providers exchange SOAP messages.

It should be mentioned, that often the service requester and service provider starts by getting to know each other before agreeing on the usage of semantics and service description [WSA04].

3.1. Semantics and Service Description

[WSA04] proposes four different ways for requesters and providers to agree on semantics and service description:

1. The requester and provider communicate directly with each other to explicitly agree on the service description and semantics.
2. The provider publishes and offers both the service description and semantics that the requester must accept unmodified as conditions of use.
3. The service description and semantics are defined and published by the requester and offered to providers as conditions of use.
4. The service description and semantics are defined as a standard by an industry or academic organization, and is used by many requesters and providers.

Communicating directly to explicitly agree on service description and semantics might be possible if there is a limited number of requesters and providers involved. In the case of price comparers and online shops there are thousands of parties involved, which make this approach unsuitable. The case that the provider or requester offers both service description and semantics would imply that either the provider has to write an interface for all price comparers or that the requester has to develop a client for all online shops. As mentioned in the previous section, both cases are insufficient. To define service de-

scription and semantics as a standard by an industry or academic organization, which many requesters and providers use, seems to be the most suitable approach for this scenario. Besides easing the integration of existing price comparers and online shops it would also ensure quick integration of future price comparers and online shops because these would not have to develop an interface from scratch. Of course, the standard interface should take best practice from major industry players, such as *Amazon* or *eBay* among others, into consideration.

Examining the Web services collections at *xmethods* and *salcentral*, along with searching *Google* for Web services, we found a limited number of Web services providing a product search interface, such as *Amazon Web Services*, *Google Web APIs*, or *eBay Developers Program*.[4] The following proposal is a combination of these Web services, advanced search options deducted from major search engines, and the search options currently available at *Elm@r*. A comprehensive comparison work including the latest development of Web services solutions in the industry will be addressed in the future. Listing 1 illustrates our proposal for some operations that provide product information.

```
public interface ShopSearchServices extends Remote {
   SearchResponse EANSearch(EANSearchRequest esr)
      throws RemoteException, SearchServicesException;
   SearchResponse ProductSearch(ProductSearchRequest psr)
      throws RemoteException, SearchServicesException;
   SearchResponse KeywordSearch(KeywordSearchRequest ksr)
      throws RemoteException, SearchServicesException;
   SearchResponse PromotionSearch(String id)
      throws RemoteException, SearchServicesException;
}
```

Listing 1. Operations for product information extraction.

`EANSearch` enables the requester to search for products by their International Article Number (EAN). Since it is possible to deduce the ISBN from the EAN and vice versa, this function can be used for an "`ISBNSearch`" too.

`ProductSearch` enables the requester to search for products by the type of product, the brand, or description.

`KeywordSearch` enables the requester to search for products using key words. The requester can determine how the keywords should be used, choosing between all, any or exact. Furthermore, the requester can input an array of words, which should be excluded.

`PromotionSearch` enables the requester to find products, which the requester should promote on his Web site. The input parameter is an `id` that uniquely identifies the requester. This method should be used only if there is an agreement between the provider and the requester. Usually, the requester would demand a fee for the promotion of products.

---

[4] http://www.xmethods.com/, http://www.salcentral.com/, http://www.google.com/, http://www.amazon.com/, http://www.google.com/apis/, http://developer.ebay.com/DevProgram/

`SearchResponse`. The main part of the `SearchResponse` is an array of `Product` objects. Furthermore, it contains cache control features from HTTP 1.1 and an expires parameter.

`SearchServicesException` contains an error code and a belonging description. In case of an exception, a SOAP fault will be send to the receiver.

The design of the WSDL document follows the *WS-I Basic Profile* [WSI04]. The goal of the *WS-I Basic Profile* is to identify those areas of Web services that cause the most problems in interoperability scenarios and to limit these in the future. The *WS-I Basic Profile* disallows the use of encoded style documents and therefore our WSDL is written in document/literal style. Inheritance is outside of the *WS-I Basic Profile*, but not directly excluded [WSI04, Br04]. Keeping in mind that Web services have the purpose of moving data from one place to another we decided not to make use of inheritance [Br04].

The granularity of operations is an important design issue. The use of coarse-grained interfaces for external consumption is recommended [WSA04, Br04]. Granularity services should use few operations with relatively large and complex messages [WSA04]. Thus, the interface operations were designed to do the complete processing of a search service.

As mentioned in Section 2, *Elm@r* uses XML standards for the exchange of product information. [VTK03] describes the importance of standardized semantics for Web services and how to use existing XML vocabulary in Web services. However, before integrating the XML into a Web service it can be useful to examine if there are parts of the XML that can be replaced by Web services functionality. For instance, the *Elm@r* XML contains an error section, which can be replaced by exception handling using SOAP faults. Listing 2 shows the error part.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<osp:ProductList
   xmlns:osp="http://elektronischer-markt.de/schema"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://elektronischer-markt.de/schema
      http://kuhlins.de/elmar/schema/products.xsd">
   <Common>
      <Version>1.1</Version>
      <Language>de</Language>
      <Currency>EUR</Currency>
   </Common>
   <Error>
      <Code>...</Code>
      <Description>...</Description>
   </Error>
</osp:ProductList>
```

Listing 2. XML response in case of an error.

[VTK03] describes two approaches for integrating existing XML into Web services: procedure-oriented and document-oriented. Here we choose the procedure-oriented

approach due to the automatic generation of code skeleton and the use of *Remote Procedure Call* (RPC).

3.2. Web Services Discovery

After having agreed on service description and semantics the service requester and service provider must become known to each other. The question is how price comparers can discover online shop services. The price comparer is the service initiator and must obtain the address of a provider. Obtaining the address of a provider can be done by getting the address directly from the provider or the requester uses a discovery mechanism to locate the service description [WSA04].

The advantage of direct discovery lies within its simplicity. Online shops can register directly by submitting their service endpoint to price comparers like *Elm@r*. The online shops avoid having to acquire technical knowledge of discovery mechanisms such as UDDI and ebXML registries.

A drawback of this approach is the lack of automation. Online shops must discover price comparers and then submit their endpoint to all price comparers manually. Finding and keeping track of price comparers is cumbersome. Submitting the endpoint to price comparers involves human interaction, which is rather time-consuming. For the price comparer the drawback lies within the lack of control. Price comparers cannot assure that all online shops will submit their service endpoint, which would decrease the price comparer's quality of service.

Discovery mechanisms contain information that allows organizations to discover and make use of services of potential business partners. A provider company submits an entry to a registry and categorizes it in ways that will make it easy to find. The service requester can search a registry for information using a wide range of criteria. There are currently two different discovery mechanisms, namely the UDDI registry and the ebXML registry [To03].

IBM, *SAP*, and *Microsoft* maintain a public production UDDI registry [To03].[5] An online shop can register at any of these companies because changes made in one UDDI registry are automatically transmitted to the other registries. In this way, the registries appear as if there was just one single, global registry [To03].

Since the above-mentioned WSDL document is the main criteria for distinguishing online shops from other organizations, it would be practical to associate the WSDL document with information about the online shop. In an UDDI registry, the WSDL interface definitions can be registered as UDDI `tModels`. The `overviewDoc` field in each new `tModel` will point to the corresponding WSDL document. The `tModels` are `wsdlSpec tModels`. Online shops can then associate the `tModel` with their `ServiceBinding` using a `tModelKey` [CER02].

---

[5] http://uddi.ibm.com/ubr/registry.html, http://udditest.sap.com/, http://uddi.microsoft.com/

```
package shop.ws;

public class ShopSearchServices_Impl
      implements ShopSearchServices, java.rmi.Remote {
   public SearchResponse EANSearch(EANSearchRequest esr)
         throws RemoteException, SearchServicesException {
      SearchResponse sr = new SearchResponse();
      // enter backend data here …
      return sr;
   }
   …
}
```

Listing 3. Automatically generated source code.

In an ebXML registry, the WSDL document can be stored as an `ExtrinsicObject`. `ExtrinsicObjects` are Meta data describing content whose type is not known to the registry [OE03]. `ExtrinsicObjects` are often used together with `SpecificationLinks`. The `SpecificationLink` provides a linkage between the `ServiceBinding` and the WSDL.

The advantage of using discovery mechanisms is that once an online shop has registered its services in a registry all price comparers can find it. The drawback of this approach is that the online shops must be acquainted with the UDDI or ebXML standards.

After having presented how price comparers and online shops agree on service description and semantics, along with how to become known to each other, it is time to describe how the price comparer and online shops uses these technologies to automatically collaborate with each other.

3.3. Online Shop Solution

In this subsection, we demonstrate the steps for online shops to set up their Web services using the *Java Web Services Developer Pack* (JWSDP)[6]. Other Web service development environments offer similar approaches.

Automatic Source Code Generation. The `wscompile` tool from the JWSDP enables online shops to automatically generate Java source files from the WSDL definition.

Integrating the Backend System. Online shops must integrate their backend systems into the generated source code. In this scenario online shops would provide the `shop.ws.SearchResponse` object with appropriate data, see Listing 3.

Compiling and Deploying the Web service. After having integrated the backend system, an online shop uses the `wscompile` and `wsdeploy` tool from the JWSDP to compile and deploy the Web service.

---

[6] http://java.sun.com/webservices/jwsdp/index.jsp

Fig. 1. Registry observer

Publishing the Web service. The final step is to publish the deployed Web service in a registry. If the online shop uses an UDDI registry, it can associate the `ServiceBinding` with the above-mentioned WSDL `tModelKey`.

3.4. Price Comparer Solution

Our price comparer solution is based on a multi-tiered architecture that is implemented using Java technologies. A user can request a product using a search form on the *Elm@r* Web site. The presentation layer communicates with a mediator that concurrently connects with remote Web services over a Web service client. The mediator is developed in accordance with the mediator framework proposed by [KK03].

Generate Client-Stubs. The Web service client uses client-stubs that are generated automatically from the WSDL file using the `wscompile` tool from the JWSDP. Because we have a WSDL definition, we use a static stub approach for developing Web services clients.

Search Registry for WSDL compatible Web services. For the mediator to know which online shops to invoke the price comparer uses a registry observer to keep track of Web services. The registry observer uses a `tModelKey` to search registries for WSDL compatible Web services.

The screenshot in Figure 1 shows the user interface of our registry observer. The administrator can set the interval in which the registry observer queries the listed registries. If the registry observer discovers a WSDL compatible Web service, it adds it to the list of new shops where it will be evaluated for further use.

```
import de.elmar.ws.generated.*;

public class SearchClient {
   private ShopSearchServices_Stub stub;
   public SearchClient(String shopUri) throws Exception {
      stub = (ShopSearchServices_Stub)
         (new ShopSearchServices_Impl()
            .getShopSearchServicesIfPort());
      stub._setProperty(
         javax.xml.rpc.Stub.ENDPOINT_ADDRESS_PROPERTY,
         shopUri);
   }
   public Product[] eanSearch(EANSearchRequest esr)
         throws Exception {
      SearchResponse sr = stub.EANSearch(
         new EANSearch(esr)).getResult();
      return sr.getProductArray();
   }
}
```

Listing 4. A Web service client using static stubs.

Invoke Web services. During a real-time search request the mediator concurrently invokes accepted shops by their service endpoints, see Listing 4.

3.5. Summary

Figure 2 summarizes the process of developing Web services for transmitting product information. The process starts with a definition of the WSDL document (1). Once the WSDL has been defined, it is published to a registry (2). Based on the WSDL, online shops develop WSDL compatible Web services with the help of compile and deployment tools, along with minor programming efforts for backend data integration (3.a). Afterwards the online shops publish their WSDL compatible Web services to a registry (4). Price comparers use the standardized WSDL to automatically build client-stubs and develop Web service clients (3.b). Searching registries for WSDL compatible Web services the price comparers attain service endpoints of online shops (5). Finally, price comparers and online shops communicate via SOAP messages (6).

4. Conclusion

The objective of this paper was to investigate the opportunities and challenges related to real-time access to product information using Web services. Moreover, the aim was to identify how to apply Web service technologies in order to attain automatic price comparisons. Besides the technical perspective, it was a question of finding a solution that appeal to organizations with modest IT budgets.

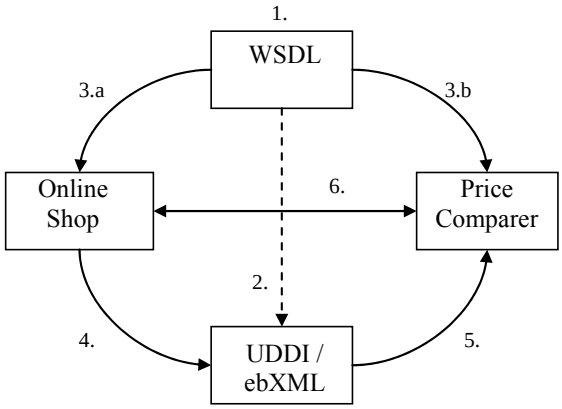

Fig. 2. The process of developing Web services for access to product information

It can be concluded that the process of creating Web services based on standard interface descriptions require a minor amount of work and is therefore even practicable for small online shops. Furthermore, it enables automatic implementation, deployment, discovery, and execution of price comparison Web services. Generally, the process can be applied in any business scenario that relies on information transmission between multiple providers and requesters, such as electronic markets, product search engines, shop directories, and so on.

Despite the opportunities of Web service technologies it will be interesting to see if they will be generally accepted in business scenarios with multiple requesters and providers. In contrast to information extraction based on wrappers, Web services for automatic information transmission demand involvement of information providers and requires suitable WSDL documents. The key challenge of Web services for automatic information transmission is therefore defining such WSDL files. Finding an industry or academic organization that will take on the challenge of defining and maintaining appropriate WSDL documents constitutes another hurdle on the path to Web services acceptance.

The authors are grateful to the three anonymous reviewers for their valuable suggestions on an earlier draft of this paper.

References

[Br04] Brown, K.: Web Services Value Type Inheritance and Interoperability, IBM, 2004. http://www-106.ibm.com/developerworks/webservices/

[CER02] Curbera, F.; Ehnebuske, D.; Rogers, D.: Using WSDL in a UDDI Registry. http://www.uddi.org/pubs/wsdlbestpractices.pdf

[Ei99] Eikvil, L.: Information Extraction from World Wide Web – A Survey, 1999. http://citeseer.ist.psu.edu/eikvil99information.html

[Ga02] Galbraith, B.; Hankison, W.; Hiotis, A.; Janakiraman, M.; Prasad, D.V.; Trivedi, R.; Whitney, D.; Motukuru, V.: Professional Web Services Security, Wrox Presss, 2002.

[HMS02] Hansen, M.; Madnick, S.; Siegel, M.: Process Aggregation Using Web Services. In: Bussleret al. (eds.): WES 2002, LNCS 2512, Springer-Verlag, Berlin, 2002; pp. 12–27.

[KK03] Kuhlins, S.; Korthaus, A.: A Multithreaded Java Framework for Information Extraction in the Context of Enterprise Application Integration. In: ISICT'03, ACM International Conference Proceedings Series, 2003; pp. 535–540.

[KT92] Kuhlins, S.; Tredwell, R.: Toolkits for Generating Wrappers – A Survey of Software Toolkits for Automated Data Extraction from Websites. In: Aksit, M.; Mezini, M.; Unland, R. (eds.): NODe 2002, LNCS 2591, Springer, Berlin, 2003; pp. 184–198.

[MSZ01] McIlraith, S.A.; Son, T.C.; Zeng, H.: Semantic Web Services. In: IEEE Intelligent Systems, Vol. 16 (2), 2001; pp. 46–53.

[OE03] OASIS/ebXML Registry Technical Committee: Registry Information Model v2.5, 2003. http://www.oasis-open.org/committees/regrep/documents/2.5/specs/ ebrim-2.5.pdf

[To03] Topley, K.: Java Web Services in a Nutshell, O'Reilley & Assosiciates, 2003.

[VTK03] Vögler, G.; Tredwell, R.; Kuhlins, S.: The Use of Existing XML Vocabularies for Web Services – Querying Product Information with Web Services and BMEcat. In: Proc. 4th Int. Conf. on Object-Oriented and Internet-based Technologies, Concepts, and Applications for a Networked World (Net.ObjectDays), Erfurt, 2003.

[WSA04] W3C: Web Services Architecture, 2004. http://www.w3.org/TR/ws-arch/

[WSI04] WS-I: Basic Profile Version, 2004. http://www.ws-i.org/Profiles/BasicProfile-1.0.html