The Use of Existing XML Vocabularies for Web Services
– Querying Product Information with Web Services and BMEcat –

Gabriel Vögler,[1] Ross Tredwell,[2] and Stefan Kuhlins[2]

[1] DaimlerChrysler AG, Research and Technology, Software Architecture (RIC/SA),
D–89013 Ulm, Germany
gabriel.voegler@daimlerchrysler.com

[2] Department of Information Systems III, University of Mannheim,
D–68131 Mannheim, Germany
{kuhlins|tredwell}@uni-mannheim.de

Abstract: Dynamic Web service applications, e.g. querying product information from different suppliers for the purpose of a price comparison, can only run automatically if, beside the technical interface, the semantics of the data are standardized. Several XML vocabularies, which, among other things, address the exchange of product data, have been developed independent of Web services (e.g. BMEcat, cXML, and xCBL, just to name a few). However, to our knowledge, these standards have hardly been mentioned in conjunction with Web services for querying product information. This article examines how BMEcat, as an example for such a standardized "business language", can be used within a Web service interface. By using specific program code examples a "procedure-oriented" and a "document-oriented" approach are compared.

1 Introduction

In a rare show of unity, the large software companies depict "Web services" as the technology of the future for the integration of heterogeneous systems on the Internet [e.g. 8, 11]. The consequent use of standards should simplify the integration of systems by allowing a high level of automation. Particularly for B2B collaboration, Web services promise new possibilities for information exchange. Such a scenario is described by IBM in a vision of "dynamic e-business" [8]. According to the Web service model in [13], a company's information system should be able to automatically find other companies that are relevant to the nature of the business transaction and connect with their information system. The Web service technology should then provide several methods to process the transaction with minimal manual intervention.

For instance, this type of concept fits in well with the idea of e-procurement. It should be possible for a company's procurement system to find all potential suppliers of a certain product and to query their electronic catalog, thus enabling the best offer to be determined. Contrary to electronic markets, which require registration, this concept takes into account all suppliers that offer the appropriate Web services.

The first technical standards to implement such a complex scenario are already available or will soon be issued. The *Simple Object Access Protocol* (SOAP) [3] defines a mechanism for the transmission of XML messages with standard Internet technologies. Furthermore, an optional SOAP module describes how to use XML to accomplish remote procedure calls (RPC). This is often combined with another optional module, which defines encoding rules for a programming language oriented type system.

The *Web Service Description Language* (WSDL) is an XML language for describing Web services [5]. At its core, it provides an XML Schema [1, 2] based syntax for the abstract description of the XML messages of Web services. Furthermore, it includes a binding mechanism, which allows the declaration of specific end-points for a selection of transmission protocols. Besides SOAP, there are also bindings for HTTP-GET/POST and MIME.

A further standard relevant in this context is called *Universal Description, Discovery and Integration* (UDDI) [16]. This is a directory service for publishing and locating Web services on the Internet. For instance, this could be helpful to find all catalog Web services of suppliers offering a certain range of products. However, this is not discussed here in detail.

In accordance with Shannon [17] the "automatic price comparison" communication problem can be solved on the syntactic or technical level with the above-mentioned Web service standards. In order to achieve a fully automatic electronic price comparison, standardization on the basis of semantics is also required. An automated comparison is only possible if suppliers' product and price information is provided in a unified data model.

To simplify the concept, we use existing XML vocabularies defined by XML Schema. An example of a vocabulary for the scenario discussed above is *BMEcat*. "BMEcat is by far the most widespread exchange standard for electronic product catalogues in German-speaking countries." [14] For instance, the BMEcat syntax contains data elements for the specification of article master data (part number, abbreviation, price, and delivery time), structural data (product group affiliation, product classification), and various additional information (e.g. keywords, references to additional multimedia data). By integrating different product classification systems, (e.g. eCl@ss [10]) products can be assigned to standardized product groups. The BMEcat authors provide a DTD as well as an XSD (XML Schema).

BMEcat is merely an example of a "business language" which is available as an XML industry standard. These standards are not promoted by the Web service idea and are often formed independently of the method of transmission.

The remainder of this paper is structured as follows: Section 2 takes a look at the SOAP and WSDL specification to demonstrate how an existing XML Schema can be integrated. At the beginning of Section 3, two different approaches are described for the XML Schema integration ("procedure-oriented" and "document-oriented"). For each approach code examples are provided for the implementation of a Web service, which queries product information from an online shop. Finally, Section 4 summarizes and draws conclusions by comparing the advantages and disadvantages of both approaches and lists possible areas for further development.

2 Integration of XML vocabularies in SOAP and WSDL

The modular aspect of the SOAP standard has to be accounted for when considering the integration of BMEcat into the SOAP protocol. SOAP can be used either as a simple transport mechanism for various XML data or as a complex RPC protocol with its own type system. In the first case, only the messaging framework (chapter 2–4 of [3]) and the HTTP Binding (chapter 6) of the SOAP standard is used. In the latter case, the SOAP messages must also comply with the RPC guidelines (chapter 7) and the SOAP encoding rules (chapter 5). Should an existing XML Schema be used for the actual SOAP message payload, this would lead to two options. Firstly, the SOAP messages contain the exact XML syntax described by the BMEcat schema. Secondly, the application of the encoding rules results in a modified XML structure, which is optimized towards RPC.

A critical question is how the BMEcat schema can be used within the WSDL description of the Web service. Fortunately, WSDL uses XML Schema to define XML messages abstractly. Furthermore, it provides an import mechanism, which allows the use of external schemas. If the schema is imported in this way, all BMEcat types can be used for the formulation of SOAP messages. Consequently, by reusing external XML Schemas, a good integration of Web services and BMEcat is made possible.

The SOAP messages defined above have to be bound to specific transmission protocols and explicit communication end-points have to be assigned. Within WSDL, this takes place in a separate module. These settings decide whether the XML Schema definition is solely the starting point for the application of certain encoding rules or includes the description of the specific transfer syntax. This refers to both types of SOAP messages mentioned above.

However, there is one restriction concerning the seemingly simple integration of BMEcat in WSDL, which needs to be addressed. The WSDL standard suggests some guidelines (chapter 2.2 in [5]), which extend or limit XML Schema (e.g. use element form instead of attributes and *ArrayOfXXX* notation [5]). These guidelines should simplify the mapping from the abstract XML Schema description to a specific representation, which complies with the SOAP encoding rules (RPC type of SOAP). This optimization in line with the SOAP encoding rules favors a programming language oriented data model. Unfortunately, the BMEcat schema does not comply with these guidelines. The possible difficulties caused by this non-conformity will be discussed later.

3 Two approaches illustrated using one example

The above examination of SOAP and WSDL results in two approaches relevant to this discussion:

- **Document-oriented approach**: Only the SOAP messaging framework is used. The SOAP body contains XML data, which does not have to comply with encoding rules. The corresponding WSDL file is configured to specify that the embedded schema describes the specific transfer syntax.

- **Procedure-oriented approach**: The SOAP messages must also comply with the RPC guidelines and the encoding rules of the second part of the SOAP standard. The corresponding WSDL file is configured to specify that the embedded schema must be interpreted as an abstract description of the messages. The specific transfer syntax results from the application of the SOAP encoding rules.

Both approaches have advantages and disadvantages depending on the specific requirements of the application. The following section analyses in more detail the differences regarding the integration of XML vocabularies (such as BMEcat) using an extended version of the above e-procurement example.

3.1 Example: Online Shop

To provide a deeper understanding, we will now discuss an online shop that offers standard possibilities for product searches. It is based on a classic multi-tier architecture, which is implemented with J2EE technologies. The data layer is implemented using a typical relational database system, the business logic uses EJB components and the presentation layer consists of JSP's. The EJB's include several search functions, which are invoked by the JSP front-end. For the following discussion the method

```
public Product[] searchByDescription(String description)
{ … }
```

is selected as an example. It enables the search for products by description (e.g. "camera"). The type `Product` is a JavaBean, which has the following methods:

```
public String getDescription() { … }
public String getProductId(){ … }
public float getPrice(){ … }
```

The updating and maintenance of the catalog is achieved by means of a supplementary software module, which is capable of exporting the whole catalog as a BMEcat file. The search capabilities of `searchByDescription()` can then be released as a Web service, whereby the `ARTICLE` model of the BMEcat standard replaces the proprietary `Product` data model.

The WSDL description is the starting point for both approaches in order to comply with the requirement to standardize the interface for the Web service. The first step is to import the BMEcat schema to enable its use for type declarations. The actual message sent for the product search is defined by a so-called WSDL procedure.

The purpose of BMEcat is to enable the interchange of whole product catalogs. No suitable syntax for querying certain products is available to date. Therefore, a valid BMEcat document must contain a list of `ARTICLE` items and some additional header fields (catalog number, name of the supplier, etc.). This supplementary information is essential, but often of no particular interest for the user of the interface. Therefore, the method `lookupDescription()` uses either dummy data or a subset of BMEcat,

depending on the approach. At this point the interface description of the two approaches differs. Consequently, they are discussed individually in the following sections.

3.2 The procedure-oriented approach

With the procedure-oriented approach, the WSDL description in the preceding section is developed further towards an RPC interface. This is achieved using a WSDL procedure `lookupDescription()` with the return type `Article[]`. The definition of this type is referenced to the corresponding type of the imported BMEcat schema. Thus, only a subset of the BMEcat vocabulary is used. At the end of the WSDL specification, the SOAP encoding rules are referenced within the binding element. (In the SOAP binding the attribute `style` has the value `rpc`.)

The fact that the application of the SOAP encoding rules simplifies the mapping between XML data and programming language types leads to an advantage for the implementation, because it allows the use of standard serializers and code generators. A so-called *Service Template Generator* can create a code skeleton for the implementation of the catalog Web service, as described in the WSDL definition. For instance, the Tool *CapeStudio* [4] produces the class structure shown in Listing 1. However, the method `lookupDescription()` must be hand coded by using already existing classes, which were previously generated.

```
public class BMEcatServerBinding
    implements BMEcatServerBindingInterface {

  public generated.bmecat.ARTICLE[]
   lookupDescription(String description) throws Exception {

    // invoke existing search implementation
    de.myCompany.Product[] products
      = de.myCompany.shopServer.searchByDescription(description)
    // construct BMEcat types
    ARTICLE[] articles = new ARTICLE[products.length];

    // fill ARTICLE with data of Product
    for (int i = 0; i < products.length; ++i) {
      articles[i] = new de.myCompany.generated.ARTICLE();
      articles[i].setSUPPLIER_AID(product[i].getProductId());
      articles[i].getARTICLE_DETAILS.setDESCRIPTION_SHORT
        (product[i].getDescription());
      articles[i].getARTICLE_PRICE_DETAILS.getARTICLE_PRICE
        .setPRICE_AMOUNT(product[i].getPrice());
      // ... more BMEcat fields
    }
    return articles;
  }
}
```

Listing 1: Implementation of the procedure-oriented approach

The class `generated.bmecat.ARTICLE` is created automatically as a Java representation of the BMEcat type `ARTICLE`. It includes some methods for the serialization into the SOAP type system. In accordance with the Bean-pattern, the data han-

dling of such types is achieved by the majority of tools using “getter” and “setter” methods. In order to make the method `lookupDescription()` work, an object of the type `ARTICLE[]` must be created and initialized with data of the corresponding `Product[]` object. This leads to the following command:

```
articles[i].setSUPPLIER_AID(product[i].getProductId());
```

This means that the mapping of the proprietary product model of `Product` to the BMEcat type `ARTICLE` requires hand coding. The developer does not have to deal directly with SOAP or XML though (see Figure 1). For this reason, the use of the BMEcat export functions is not recommended, because the Service Template Generator only works with Java data types.

In the approach described above, the BMEcat model can be used at a semantic level in conjunction with an RPC-based Web service. Unfortunately, this is only true for the abstract specification of the Web service. Due to the application of the SOAP encoding rules, the Web service does not use the original syntax defined in the BMEcat schema. Instead, an adapted version is implemented, which uses the SOAP type system.

The direct integration of the BMEcat schema leads to problems with *CapeStudio*. The tool generates code for the class `generated.bmecat.ARTICLE` that cannot be compiled. It seems that it requires a standard WSDL description that complies with the recommended conventions for XML Schema.

3.3 The document-oriented approach

In the case of the document-oriented approach, the WSDL definition in Section 3.1 is extended to enable the exchange of XML documents. With the WSDL procedure `lookupDescription()` a similar query interface to the RPC example is defined. Here, the corresponding response message has the type `BMECAT`. This means that the response entails a complete catalog structure that contains more information than is required for the purpose of price comparisons. Such an interface design has the advantage of easier integration with other BMEcat applications. Furthermore, the BMEcat schema only defines a few mandatory fields that are irrelevant in the context of `lookupDescription()`.

Encoding rules or RPC guidelines are not specified in the SOAP binding. (The attribute `style` has the value `document`.) Hence, the SOAP message payload can be validated against the BMEcat schema.

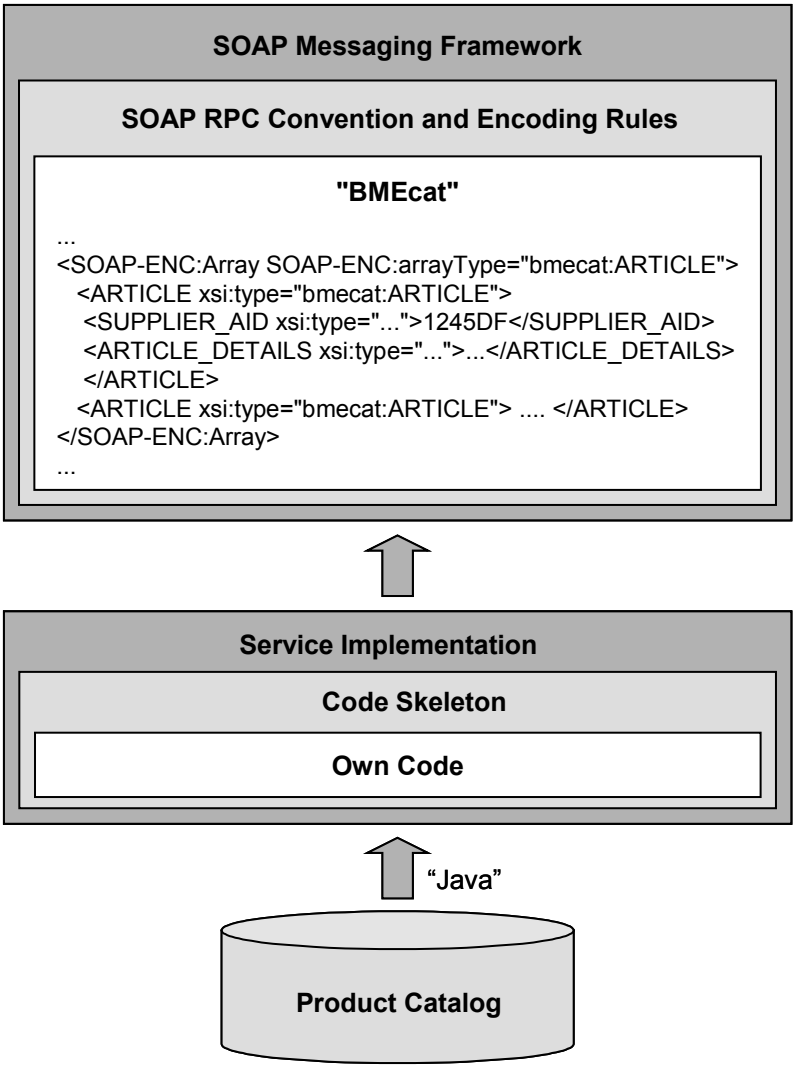

Figure 1: The procedure-oriented approach

The use of arbitrary XML, and the absence of the SOAP encoding rules, makes the process of serializing and deserializing between Java and XML more complicated than in the case of the procedure-oriented approach. At this point, so-called *XML-Binding-Frameworks* (e.g., JAXB [12]) could be a solution, which allow the generation of XML serializers for Java classes. However, such technologies are not integrated in the Web service tools mentioned in this article. Therefore, a fully automatic Web service generator, as in the case of the procedure-oriented approach, is not available.

The abandonment of SOAP encapsulating code requires more SOAP specific knowledge and manual coding. For instance, one option for this scenario is the JAXM framework [15]. On the one hand, it supports the developer with a standard Servlet enabling own service implementations based on derivation. On the other hand, it provides a SOAP API for the generation and handling of SOAP messages. In addition to the basic structure of a SOAP message (`SOAPMessage`, `SOAPHeader`, `SOAPBody`) the API also allows the handling of the XML relevant data of the SOAP-Body. Similar to the procedure-oriented solution, the existing Java method `searchByDescription()` could be used to successfully insert XML nodes of the type `ARTICLE`. The following code extract should indicate the extent of the required detail:

```
SOAPElement ARTICLE = newCatalog.addChildElement("ARTICLE");
SOAPElement ARTICLE_DETAILS
  = ARTICLE.addChildElement("ARTICLE_DETAILS");
SOAPElement DESCRIPTION_SHORT
  = ARTICLE_DETAILS.addChildElement("DESCRIPTION_SHORT");
DESCRIPTION_SHORT.addTextNode(products[i].getDescription);
```

It becomes clear that this approach is quite cumbersome. For this reason, it is better to use the BMEcat interface of the catalog software instead of the existing Java implementation. This means that the search function of `searchByDescription()` must be re-implemented with XPath [7] expressions. The advantage derived is an end-to-end XML solution, which requires no mapping between different data representations (e.g., Java to XML and vice versa). The result of such a query is already in BMEcat format and can be directly inserted into the SOAP body. Unfortunately, the SOAP API of JAXM does not use a DOM-based object model. For that reason, inserting DOM nodes into the SOAP body is quite difficult. To use existing XML for SOAP messages, JAXM requires the whole message (including the header) as a DOM document. However, a complete manual generation of SOAP messages via DOM means time consuming low-level programming.

```
import javax.servlet.*;
import javax.xml.messaging.*;
import javax.xml.soap.*;
import javax.xml.parsers.*;
import org.w3c.dom.*;
import org.apache.xpath.*;

public class QueryListener extends JAXMServlet
    implements ReqRespListener {

  /* The initialization of the following objects is omitted:
   * bmeCatDocument: complete catalog of the shop as DOM Document
   * bmecatTemplate: catalog template for the response message
   *   (contains no articles, but dummy data for mandatory fields)
   * messageFactory: Factory for the generation of SOAP messages
   */

  public SOAPMessage onMessage(SOAPMessage inMessage)
      throws Exception {
```

```
    // extract search string from SOAP request
    SOAPElement inBody
      = inMessage.getSOAPPart().getEnvelope().getBody();
    String description = inBody.getFirstChildElement().getValue();

    // process XPath query
    org.w3c.dom.NodeList resultList =
     XPathAPI.selectNodes(bmeCatDocument,
      "//ARTICLE[contains(ARTICLE_DETAILS/DESCRIPTION_SHORT, "
      + description + ")]");

    // generate BMEcat document for the response
    org.w3c.dom.Document bmecatResult =
     (Document) bmecatTemplate.cloneNode(true);

    // fill bmecatResult with resultList's ARTICLE elements
    // ...

    // generate SOAP response
    SOAPMessage outMessage = messageFactory.createMessage();
    SOAPBody soapBody
      = outMessage.getSOAPPart().getEnvelope().getBody();

    insertDOM(soapBody, bmecatResult);
    return outMessage;
  }

  private void insertDOM(SOAPBody soapBody,
      org.w3c.dom.Document document) throws Exception {
    // inserts a DOM document into the SOAP body
    // ...
  }
```

Listing 2: Implementation of the document-oriented approach

Listing 2 shows an implementation of the Web service, which uses techniques described by [9] to insert DOM nodes into the SOAP body. As a result, the code only contains instructions for the receipt of a message, the invocation of the search method, and the generation of the response message. For the latter step, the code can be limited to the following instruction:

```
insertDOM(soapBody, bmecatResult);
```

A continuous XML solution is achieved reusing XML documents generated by the catalog system for the SOAP messages. No complex data serialization is necessary (see Figure 2). The procedure-oriented approach requires manual coding for the mapping between `Product` and `ARTICLE`. In comparison, the document-oriented approach requires less development effort for the shop example. One disadvantage is the fact that SOAP messages have to be explicitly generated at the XML level. Therefore, developers need more detailed knowledge of the SOAP protocol and the JAXM API.

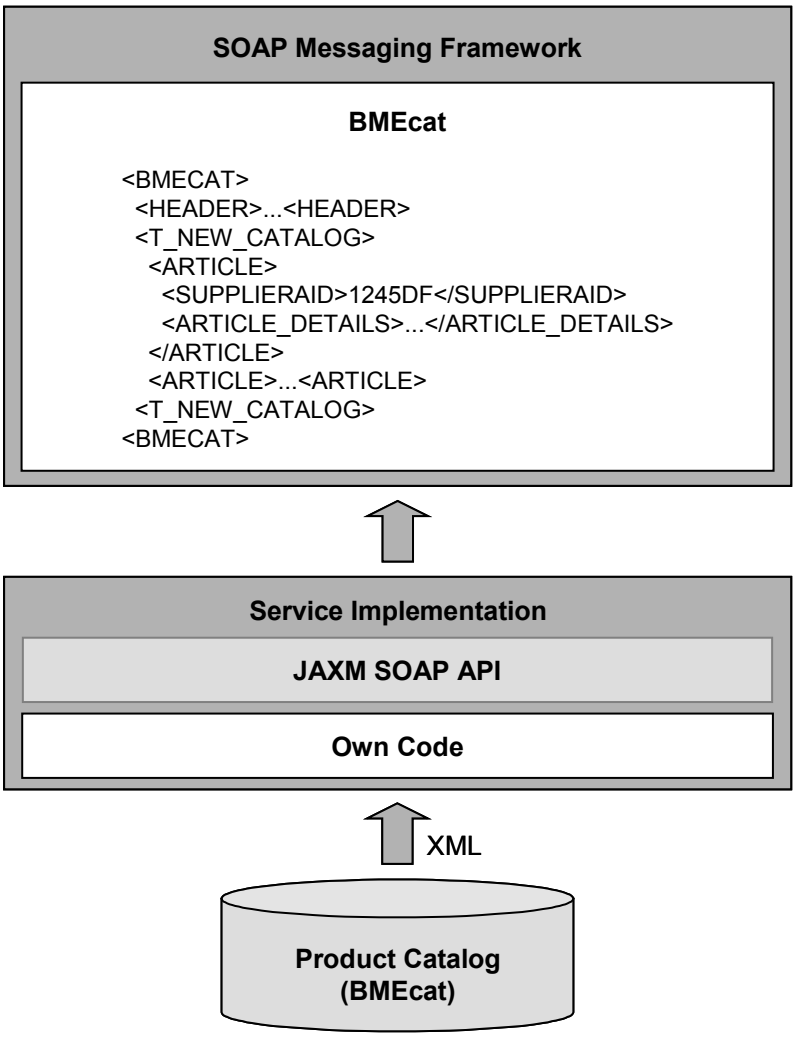

Figure 2: The document-oriented approach

4 Conclusion and Future Directions

The importance of standardized semantics for dynamic integration applications has been emphasized by the example of requesting and delivering product information in the context of comparing offers. The possibility of using the existing XML industry standard BMEcat within a Web service was examined. Two approaches were discussed: a document-oriented and a procedure-oriented approach.

The procedure-oriented approach appears very implementation-biased, because the RPC conventions and the encoding rules allow the integration of a Web service with standard elements of a programming language. The use of XML is hidden from the developer, because the type conversion is done automatically. However, this means that XML technologies for transformation (e.g., XSLT [6]), validation (e.g., XML Schema) or data navigation (e.g., XPath) are not directly available. For instance, the type constraints of the BMEcat schema are lost during type conversion, because the Java classes generated by CapeStudio do not contain corresponding code for checking data integrity.

The document-oriented approach is more focused on the exchange of XML documents. The structure of these documents is generally determined by the underlying information and is not optimized for implementation specific aspects. The Web service must be programmed at the XML level, which, on the one hand requires specific API knowledge, but on the other hand allows the utilization of powerful XML technologies. Furthermore, the product data keeps its original structure, so that it can be validated against the BMEcat schema.

In conclusion, it can be said, that the document-oriented approach is basically a better method to resolve the problem of XML-vocabulary integration for Web services, because it has less restrictions regarding the utilization of XML. The programming language bias of the procedure-oriented approach complicates the integration of XML industry standards into Web services.

The claims of some software companies that their tools can expose legacy applications as Web services, without the need for XML knowledge, should be viewed critically. The example discussed in this article shows that the vision of "dynamic e-business" is only feasible if Web services offer their data in a standardized semantic form. The advertised fully automatic tools only partially fulfill this prerequisite. The favored document-oriented approach requires some XML programming, but it offers the benefit of a completely standardized interface.

To increase interoperability, the standardization on the semantic level should be intensified in the future. Unfortunately, BMEcat is only used in German-speaking countries. A worldwide accepted standard for product data is not yet available. Furthermore, the committees responsible for standardization must provide "Web service bindings" for their XML vocabularies. This would lead to a consistent use of a specific vocabulary within a Web service, which is an important prerequisite for the realization of the fully automatic price comparison.

As mentioned, BMEcat does not provide a syntax for querying product information. Therefore, a standard query interface should be specified, which defines the possible search criteria. In addition, an appropriate result structure must be standardized, which excludes unnecessary data. This would prevent the use of dummy data, as illustrated in our example.

References

1. Beech, D.; Lawrence, S.; Maloney, M.; Mendelsohn, N.; Thompson, H. S. (eds.): XML Schema Part 1: Structures. World Wide Web Consortium, Boston, USA, 2001. W3C Recommendation. URL: http://www.w3.org/TR/2001/REC-xmlschema-1-20010502/
2. Biron, P. V.; Malhotra, A. (eds.): XML Schema Part 2: Datatypes. World Wide Web Consortium, Boston, USA, 2001. W3C Recommendation. URL: http://www.w3.org/TR/2001/REC-xmlschema-2-20010502/
3. Box, D.; Ehnebuske, D.; Kakivaya, G.; Mendelsohn, N.; Frystyk Nielsen, H.; Thatte, S.; Winer, D. (eds.): Simple Object Access Protocol (SOAP) 1.1. World Wide Web Consortium, Boston, USA, 2000. W3C Note. URL: http://www.w3.org/TR/SOAP/
4. CapeClear (editor): CapeStudio 3 Technical Overview. CapeClear, USA, 2002. URL: http://www.capeclear.com/products/whitepapers/CSTechnicalOverview.pdf
5. Christensen, E.; Curbera, F.; Meredith, G.; Weerawarana, S. (eds.): Web Services Description Language (WSDL) 1.1. World Wide Web Consortium, Boston, USA, 2001. W3C Note. URL: http://www.w3.org/TR/wsdl
6. Clark, J. (editor): XSL Transformations (XSLT) Version 1.0. World Wide Web Consortium, USA, 1999. W3C Recommendation. URL: http://www.w3c.org/TR/xslt
7. Clark, J.; DeRose, S. (eds.): XML Path Language (XPath). World Wide Web Consortium, Boston, USA, 1999. W3C Recommendation. URL: http://www.w3.org/TR/1999/REC-xpath-19991116
8. Colan, M.: Dynamic e-business: Using Web services to transform business. IBM, USA, 2001.URL: http://www-3.ibm.com/software/solutions/webservices/pdf/wsintro.pdf
9. Cope, M.: How to Add a DOM Document Object as a SOAP Body Element in a SOAP Message Using Java APIs for XML Messaging (JAXM). Sun Microsystems, USA, 2002. Technical Article. URL: http://access1.sun.com/techarticles/SOAP_DOM/SOAP_DOM.html
10. eCl@ass e.V. (editor): eCl@ss – Whitepaper 0.6. eCl@ss e.V., Köln, Germany, 2000. URL: http://www.eclass.de/informationen/download/eClassWhitePaper06.doc
11. Ewald, T.: Understanding XML Web Services – The Web Services Idea. Microsoft Corporation, USA, 2002. URL: http://msdn.microsoft.com/webservices/understanding/readme/default.aspx
12. Fialli, J.; Vajjhala, S.; (eds.): The Java Architecture for XML Binding (JAXB) – Final V1.0. Sun Microsystems, Santa Clara, USA, 2003. URL: http://java.sun.com/xml/jaxb/
13. Heather, K.: Web Services Conceptual Architecture (WSCA 1.0), IBM Software Group, USA, 2001. pp. 7–9. URL: http://www-3.ibm.com/software/solutions/webservices/pdf/WSCA.pdf
14. Hümpel, C.; Kelkar, O.; Pastoors, T.; Renner, T.; Schmitz, V.: Spezifikation BMEcat Version 1.2. Fraunhofer IAO, University of Essen BLI, 2001, URL: http://www.bmecat.org/
15. Kassem, N.; Mordani, R; Vijendran, A.; Java API for XML Messaging (JAXM) v1.1 Specification, Sun Microsystems, Santa Clara, USA, 2002. URL: http://java.sun.com/xml/downloads/jaxm.html
16. OASIS – Organization for the Advancement of Structured Information Standards (editor): UDDI Version 3 Specifications, USA, 2002. URL: http://www.oasis-open.org/committees/uddi-spec/tcspecs.shtml - uddiv3
17. Shannon, C. E.: A Mathematical Theory of Communication. Bell Systems Technical Journal, Vol. 27, USA, 1948, pp. 379–423 and 623–656.