Dirk Habich, Steffen Preissler, Wolfgang Lehner, Sebastian Richly, Uwe Assmann, Mike Grasselt, Albert Maier

Data-Grey-BoxWeb Services in Data-Centric Environments

Erstveröffentlichung in / First published in:

IEEE International Conference on Web Services (ICWS 2007). Salt Lake City, 09-13.07.2007. IEEE, S. 976-983. ISBN 0-7695-2924-0

DOI: https://doi.org/10.1109/ICWS.2007.69

Final edited form was published in "IEEE International Conference on Web Services (ICWS 2007), Salt Lake City, 2007". ISBN0-7695-2924-0.
https://doi.org/10.1109/ICWS.2007.69

Data-Grey-Box Web Services in Data-Centric Environments

Dirk Habich, Steffen Preissler
Wolfgang Lehner
Dresden University of Technology
Database Technology Group
dbgroup@mail.inf.tu-dresden.de

Sebastian Richly,
Uwe Assmann
Dresden University of Technology
Software Technology Group
{sebastian.richly,ua1}@inf.tu-dresden.de

Mike Grasselt,
Albert Maier
IBM Böblingen
Germany
{grasselt,amaier}@de.ibm.com

Abstract

In data-centric environments, for example, in the field of scientific computing, the transmission of large amount of structured data to Web services is required. In service-oriented environments (SOA), the Simple Object Access Protocol (SOAP) is commonly used as the main transport protocol. However, the resulting 'by value' data transmission approach is not efficiently applicable in data-centric environments. One challenging bottleneck of SOAP arises from the XML serialization and deserialization when processing large SOAP messages. In this paper, we present an extended Web service framework which explicitly considers the data aspects of functional Web services. Aside from the possibility to integrate specialized data transfer methods in SOA, this framework allows the efficient and scalable data handling and processing within Web services. In this case, we combine the advantages of the functional perspective (SOA) and the data perspective to efficiently support data-centric environments.

1. Introduction

Web services (WS) are independent software components exposed on the Web, supporting the interoperable machine-to-machine interaction via Internet. Fundamentally, Web services are considered black-box components, since they do not offer any information on how they work; they only expose information on the structure of parameters or data they expect as input and return as result. The advantages of Web services are their interoperability, the XML-based self-descriptive interfaces and the seamless usage with established Internet protocols such as the HTTP protocol.

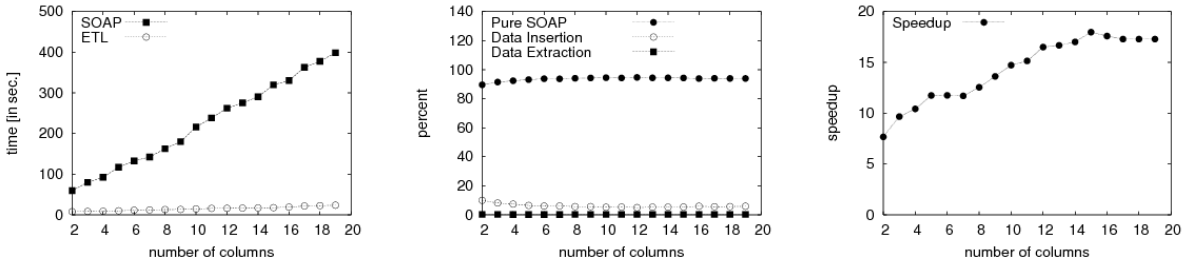
In service-oriented environments, the *Simple Object Access Protocol* (SOAP) is commonly used for the communication with and between Web services. According to the client/server paradigm, clients (service requestors) perform a Web service invocation by passing SOAP messages to a server hosting the corresponding service. These messages include a reference to the target service to invoke as well as any number of parameters and data to be transmitted to the service. Reply messages may be transmitted either synchronously or asynchronously from the server back to the client. The SOAP protocol defines an XML-based format for the messages to be used in a Web service invocation.

Various application areas, like business-to-business communication, enterprise application integration, e-Science or service-oriented computing, have implemented Web services. In many of those application scenarios, an exchange of large amounts of structured data between Web services is required. Embedding such massive structured data sets in SOAP messages is possible but not a suitable solution from the performance perspective.

Consider the following example: A service requestor manages its own massive data sets in a local data management system, e.g. in a relational database system. On a subset of these data, an analysis function should be applied and a Web service should offer the required analysis function. To use such a Web service, the service requestor has to extract the data from the data management system in order to be able to transfer them in XML-marshaled SOAP messages to the Web service. The Web service deserializes the incoming SOAP messages and inserts the contained massive data set into the local database system before the processing starts. This procedure is typical in data-centric environments, since it cannot be taken for granted that the received data can be managed in the main memory during the whole processing time.

Based on this example, we investigated the transfer time of the SOAP approach between the service requestor and the Web service. In this experiment, we transmitted $50,000$ tuples with different numbers of columns, where each column included string values of length 60. The measured transfer times includes the times for (1) the data extraction from the database system, (2) the transfer of the SOAP message from the requestor to the service, and (3) the insertion of the received data into the database system. Figure 1(a) depicts the transfer times for the different numbers of columns. As we can see, the transfer time increases with the complexity

Final edited form was published in "IEEE International Conference on Web Services (ICWS 2007), Salt Lake City, 2007". ISBN0-7695-2924-0.
https://doi.org/10.1109/ICWS.2007.69



(a) Times (b) Distribution (c) Speedup

Figure 1. Data Propagation Evaluation - SOAP versus ETL.

of the data structure (more columns). The distribution over the essential transfer steps is depicted in Figure 1(b). As illustrated, the pure SOAP transfer, including XML serialization and deserialization at the requestor and the service, causes up to 90%. This experiment confirms the following well-known drawbacks of SOAP:

1. When the data, and thus the SOAP message structure, get more complex, the performance of the SOAP transmission gradually gets worse compared to large messages with simple structures [3, 13, 16].

2. The process of XML serialization and deserialization represents a bottleneck when processing large SOAP messages, since the XML processing is a time- and main-memory-consuming process [3, 13, 16].

If the Web service exposed that a database system is used, the service requestor could apply more specialized, and therefore more efficient, data propagation tools, like ETL or replication. In the database area, the *Extract-Transform-Load (ETL)* system [17] is such a specific approach. ETL systems are designed to extract data from various systems, apply transformation rules on data and load those transformed data into a different database system. In our experiment, we used an ETL system for the exchange of data between the database systems of the service requestor and the Web service. Figure 1(a) depicts the resulting transmission times. As Figure 1(c) illustrates, we obtain a transmission speedup of up to 16 compared to the SOAP approach.

Since the performance is an important factor in many application areas, an integration of such data-specialized propagation approaches would be beneficial. In this case, the following challenges arise: (i) creation of data-grey-box Web services offering more information on the data persistence aspect within the service and (ii) the integration or triggering of data propagation tools in the Web service invocation process.

Our Contribution: In this paper, we present an extended Web service framework which explicitly considers the data aspects of functional Web services. Aside from the possibility to integrate specialized data transfer methods in Web service environments, this framework allows the efficient and scalable data handling within Web services.

Our description focuses on Web services with a dynamic data aspect. Such services receive massive data sets on which they apply the provided functionality. Furthermore, they may return a large amount of structured data. In detail, we discuss the following points:

1. We introduce a *data framework* that will be responsible for the handling of massive data sets within Web service environment. This concept oversees the complete management of the data aspect.

2. Based on the first concept, we enhance the Web service interface with special properties for the transfer of large amounts of structured data. The new Web service interface offers all necessary data-related information.

3. Finally, we present the interaction between Web service invocations and specialized data propagation tools. In this paper, we focus on database technologies.

The remainder of the paper is structured as follows: In the following section, we review related work to distinguish our work from previous research activities. Our new data framework is introduced in Section 3, followed by a description of our new Web service interface and the invocation process in Section 4. A detailed performance evaluation is described in Section 5. We conclude our paper with a summary and an outlook on future work.

2. Related Work

Recently, the performance of the SOAP protocol has received a lot of attention from the research community as well as from the industry. One prominent technique to enhance the performance is caching at the client site, server site or in dedicated hardware components [1]. Other techniques try to reduce the network bandwidth requirement by using compression [2], Binary XML or binary metadata [18]. All applied compression techniques, like gZip, XMill, and Millau [6], produce high compression ratios. In [13], a *Table-Driven XML approach (TDXML)* is proposed offering

Final edited form was published in "IEEE International Conference on Web Services (ICWS 2007), Salt Lake City, 2007". ISBN0-7695-2924-0.
https://doi.org/10.1109/ICWS.2007.69

a more compact message size, a simpler message structure and easier access to individual elements when compared to SOAP. The drawback of such methods in data-centric environments is that structured data is exchanged on a functional application level instead of triggering a data-level transfer method.

SOAP messages with Attachments (SwA) are another possibility to avoid the inefficiency of XML, i.e. requiring more bandwidth, more storage and more processing power than equivalent binary implementations. There are two approaches for SwA: MIME (Multipurpose Internet Mail Extension) and DIME (Direct Internet Message Encapsulation). Both of them define an abstract model for SOAP attachments and the mechanism for encapsulating a SOAP message and zero or more attachments in either MIME or DIME messages. Attachments are applied to the encapsulation of binary data in the form of image files and encapsulation of other XML documents as well as XML fragments [8]. An application to structured data is not practical, since all schema specifications will be lost.

Moreover, Patcas et al. [14] integrated a Data-Flow Distribution Protocol for Web Services (DFDP-WS) into the Web service stack. Their application area is the Business Process Execution Language for Web services (WSBPEL) and they want to establish a decentralized data-flow model. The proposed protocol is used to encapsulate both control-flow information and data in the messages it carries. The focus is similar to us, but the data propagtion is still done with an SOAP-like protocol approach, while we present an flexible approach to integrate various existing data propagtion tools.

The problem of data access and data sharing accross various Web services has also been examined in the Grid approach. In this special environment, the OGSA-DAI framework has been established to produce common middleware allowing uniform access to data resources using a service-based architecture [12]. The produced Grid Data Services allow consumers to discover the properties of structured data stores and to access their contents. Aside from the access to structured data, the OGSA-DAI framework offers the possibility of moving data between services with gridFTP [12].

Moreover, the database research community has paid a lot of attention to the field of data exchange between different database systems. A well-known method is the ETL (Extract-Transform-Load) approach, loading data from different data sources into a common data warehouse [17]. Such ETL processes consist of three parts: (1) extraction of data from the different source systems, (2) application of a series of rules and functions to the extracted data to derive the data to be loaded, and (3) loading of the data into a data warehouse system. This ETL approach is a data-spezialized technique to efficiently transmit structured data to various different data management systems, e.g. relational or XML database systems. A further popular data propagation method is replication [10]. In database systems, this is used to provide redundancy or to balance the load across multiple database servers.

3. Data Framework within Web Services

As already mentioned, we focus on Web services with a dynamic data aspect here. Such services are characterized by receiving massive structured data sets with additional functional parameters, applying the provided functionality and returning the result to the service requestor. This result can also be a large amount of data. Typical examples are analysis services, e.g. from the data mining area. As our ongoing example, we use a Web service implementing the k-means [4] clustering algorithm. This is an algorithm to cluster objects into k partitions based on attributes. As input, the function expects a number of data points (n-dimensional) and a parameter k. The Web service returns as result the data points with an additional attribute indicating if data points belong to the same partition.

Such data-intensive Web services typically require a local database system for the processing of the received data, since it cannot be taken for granted that all data can be managed in the main memory during the whole processing time. A further advantage of using a database system is that such systems usually offer efficient methods for the processing of massive data sets, which can be exploited then [9].

An important general property of Web services is that for each invocation a new instance will be created, and therefore, several Web service instances can run in parallel. Having regard to that we use a database, the dynamic data aspect has to be put in correlation with the corresponding Web service instance, which then affects the handling of the database system.

3.1. Framework Description

In order to allow the efficient and scalable handling of both input and output data within Web services in a standardized way, we introduce our *data framework* for Web services. We still assume that the data exchange with the service requestor is done via SOAP messages. Our framework consists of the following two essential components:

1. An abstract, object-oriented access interface, which allows unified access to different data sources, such as relational databases, XML databases or even CSV and XML files.

2. A coordinator responsible for the management of the access interface and the available database systems. In general, the coordinator is the central control instance.

We realize the abstract access interface with the help of the Service Data Objects technology (SDO) [15]. The SDO technology is an object-based specification for unified access to different data sources, like databases, files or EJBs.

Final edited form was published in "IEEE International Conference on Web Services (ICWS 2007), Salt Lake City, 2007". ISBN0-7695-2924-0.
https://doi.org/10.1109/ICWS.2007.69

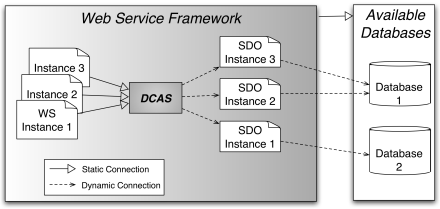


Figure 2. Data Framework.

In this paradigm, an application works on a data graph containing one or more data objects. This data graph is a tree-like connection of concrete data object instances. SDO mediators represent the mappings of the data graphs to various data sources. In order to do so, a variety of techniques may be employed, e.g. EJB or JDBC, allowing flexible storage strategies.

Aside from this abstract data access interface, we introduce a coordinator (*Data Container Administrator Service*, DCAS), which is responsible for the handling of the complete management of the data aspect. The management tasks cover: (i) the correlation of the data with the Web service instance, (ii) the appropriate preparation of the data source, e.g. the creation of tables based on the input and output schemas, and (iii) the dynamic instantiation of the SDO mediators. The DCAS is implemented as a singleton [5], i.e. there exists only one instance of DCAS and all WS instances access this one instance. Furthermore, this controlling component allows the use of extended management activities, such as load balancing over a number of data sources. Such load balancing avoids potential bottlenecks at a certain data source. These bottlenecks might result from a lot of parallel service calls by requestors, which then again result in the demand for too many parallel read or write operations or even in a lack of storage space at a data source.

Figure 2 shows our framework with three WS instances. As can be seen in the figure, there are two static connections. The first static connection represents the access to the coordinator, whereas the second static link is for the allocation of the databases to the WS framework. The instantiation of the SDOs for each WS instance and the assignment to a database are conducted dynamically at runtime.

3.2. Application Details

The DCAS and the SDOs are the central components of our framework. Obviously, the resulting explicit separation of the functional aspect from the data aspect affects the way the Web services are implemented. Figure 3 illustrates our realization approach based on the application server JBoss as WS runtime environment. First, we expect that a separate XML file exists, containing general access information of available and useable databases.

Moreover, for each function provided by a Web service, the respective schema specification for input and output data

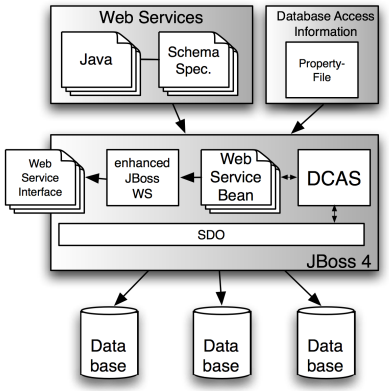


Figure 3. Realization Approach with JBoss.

is available explicitly, e.g. in a separate XML file or in the form of annotations. The services are implemented as stateless session beans using the inferface of the DCAS offering the following functionalities:

- *String:getSession(String functionID), closeSession(String sessionID)*: The correlation of the data with the WS instance is implemented through a session concept. When generating the session, the DCAS has to be informed of the function of the service wishing to be executed. This is realized by handing over the *functionID* (unique identifier) when creating the session. During this call, the DCAS executes all essential steps on the currently optimal database. For example, all tables for input and output data of the function are created according to the schema specifications. The method *getSession* returns a *sessionID* as unique identifier. After the processing and loading of the output data from the database, which is necessary to send the result via SOAP messages, the session can be closed and the database can be cleaned up.

- *getInput(String sessionID), getOutput(String sessionID)*: With the help of these two DCAS methods, the WS function gets access to the respective SDO data graphs and can thus process the data.

During the deployment of new Web services, the schema specification for each provided function is registered at the DCAS. This registration procedure is essential to allow that the SDOs can be generated and instantiated at runtime using concepts from the Model-Driven Architecture [7]. This means that programmers do not have to worry about the data aspect any longer; instead, this will be taken care of by the coordinator. Figure 4 shows the usage of our framework on the k-means function.

To summarize, we have implemented an explicit separation of the functional aspect from the data aspect within

Final edited form was published in "IEEE International Conference on Web Services (ICWS 2007), Salt Lake City, 2007". ISBN0-7695-2924-0.
https://doi.org/10.1109/ICWS.2007.69

```
public outData kmeans(int k, inData in)
{
      String sID = DCAS.createSession(UUID);
      SDO input = DCAS.getInputContainer(sID);
      SDO output = DCAS.getOutputContainer(sID);
      Transform input data „in“
        into output data „out“ (Algorithm);
      returnData = (outData)output.getData();
      DCAS.closeSession(sID);
      return returnData;
}
```

Figure 4. Using our Data Framework.

Web services through our data framework. This separation is now the perfect foundation to integrate specialized data propagation tools in the invocation procedure of Web services. Without such a data framework, the integration would be much more difficult.

4. Data-Grey-Box Web Services

In the previous section, we realized the separation of the functional aspect from the data aspect within Web services through the introduction of our data framework. So far, this separation has been completely transparent for the service requestor. As the experiment in Section 1 illustrated, the data exchange with specialized data propagation tools is more efficient than with SOAP messages.

In this section, we extend the separation of the functional aspect from the data aspect regarding the interface description in order to get data-grey-box Web services. The goals of these data-grey-box Web services are listed below:

1. Web services are considered stateless services; data-grey-box Web services will still be stateless.
2. The data exchange between service requestors and Web services shall be transparent for both sides.
3. The approach shall be flexible regarding the seamless integration of further data propagation tools.

4.1. Interface Description

In a first step, we outline our modifications of the interface description of Web services. The Web Service Description Language (WSDL) is a language based on XML which consists of two main components. In the abstract part of the WDSL description of a service, its functionalities are defined. With the help of the elements *operation*, *message* and *type* in abstract form, the WSDL *portType* defines what kind of functionalities a service provides via what kind of interfaces. For the use of the service, it is subsequently important to know how the operations can be called. In the concrete part of the WSDL description, the binding (WSDL element *binding*) for the abstract service descriptions is performed. Protocols and transfer formats calling mechanisms of the operations as well as employed message formats are defined. Finally, the Web service *endpoint* is specified, where the service is available with the described protocols and calling mechanisms.

```
<!-- parameter message for k -->
<wsdl:message_parameter name="k">
      <wsdl:part element="impl:k"/>
</wsdl:message_parameter>
<!-- data messages -->
<wsdl:message_data name="InputDataKMeans">
      <wsdl:part element="impl:InputDataKMeans"/>
</wsdl:message_data>
<wsdl:message_data name="OutputDataKMeans">
      <wsdl:part element="impl:OutputDataKMeans"/>
</wsdl:message_data>
```

Figure 5. Message Definitions.

The first measure in our attempt to separate the data aspect from the functional aspect is that we split the previously used unified message concept into messages for data and messages for parameters. In this case, the service requestor is able to differentiate between data and parameters for the individual functions of the Web service. Thus, the general construct *wsdl:message* is split into *wsdl:message_parameter* and *wsdl:message_data*, without any further changes of the actual type definitions. The message definitions for the parameters are implemented as usual, while the message definitions for the input and output data are generated based on the available schema information. Figure 5 shows the message definitions for our k-means Web service example. The function expects the parameter k for the number of clusters as input. Furthermore, there are two different schemas for data, and on this basis, messages for input and output data are specified.

After this differentiation between messages in data messages and parameter messages, the next step is to adjust the abstract definition of the individual operations, as shown in Figure 6. An operation can now receive either a parameter message or a data message as input. The same applies to the output. From this abstract description, the service requestor can learn what the operation considers parameters and what it sees as data and how the schemas look like.

In the concrete part of the WSDL description, the binding for the abstract service descriptions is realized by defin-

```
<wsdl:portType name="KMeansPortType">
      <!-- function KMeans -->
      <wsdl:operation name="KMeans">
         <wsdl:input_parameter message_parameter="impl:k"
                               name="k"/>
         <wsdl:input_data message_data="impl:InputDataKMeans"
                  name="InputDataKMeans"
         <wsdl:output_data message_data="impl:OutputDataKMeans"
                   name="OutputDataKMeans"/>
      </wsdl:operation>
</wsdl:portType>
```

Figure 6. Abstract Definition of Operations.

Final edited form was published in "IEEE International Conference on Web Services (ICWS 2007), Salt Lake City, 2007". ISBN0-7695-2924-0.
https://doi.org/10.1109/ICWS.2007.69

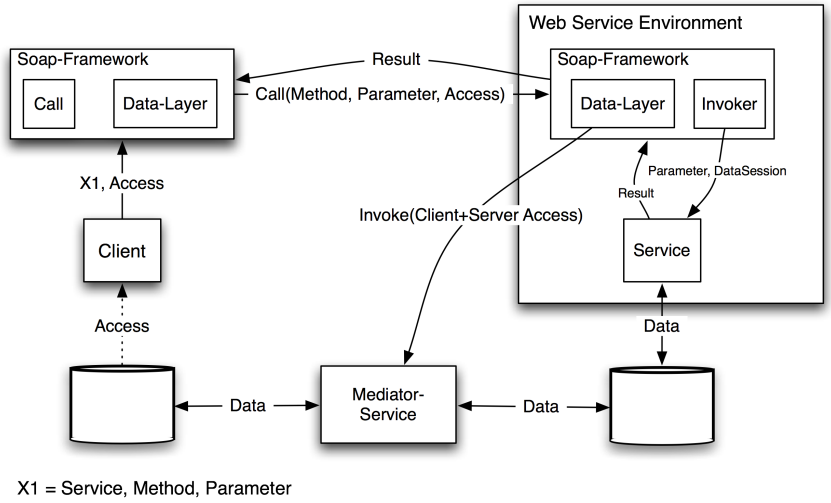


Figure 8. Data-Grey-Box Web Service Invocation Process.

```
<wsdl:binding name="KMeansServiceSoapBinding"
                      type="impl:KMeansService">
   <wsdlsoap:binding style="document"
                    transport="http://schemas.xmlsoap.org/soap/http"
                    datalayer="http://org.tud.dbs/datalayer"/>
   <!-- function KMeans -->
   <wsdl:operation name="KMeans">
      <wsdlsoap:operation soapAction=""/>
      <wsdl:input_parameter name="k">
            <wsdlsoap:body use="literal"/>
      </wsdl:input>
      <wsdl:input_data name="InputDataKMeans">
        <wsdl:datalayer use="database"/>
      <wsdl:output_data name="OutputDataKMeans">
        <wsdl:datalayer use="database"/>
  </wsdl:operation>
</wsdl:binding>
```

Figure 7. Binding for Operations.

ing protocols and transfer formats. As before, the parameter messages are handed over via SOAP when calling the function, and thus, there are no changes in this regard. For the data messages, a new transfer format *data-layer* is introduced, as shown in Figure 7. Through this new transfer format, the operation signals that the data are not to be transferred via SOAP but that there is a separate data layer instead. In this binding description, the Web service offers information on the used data management system within the Web service. In our example, the operation uses a database system (*use="database"*) for the handling of the data.

However, these structural modifications of the interface description are not sufficient. Next, we describe the invocation process for Web services.

4.2. Invocation Process

Before we are able to describe the entire WS invocation process for our data-grey-box Web services, we take a look at possible data transfer methods. Fundamentally, there are three different principles for the initiation, and hence for the execution, of the data exchange. With the first approach – *the push approach* – the service requestor is responsible for (i) sending the input data to and (ii) acquiring the output data from the Web service. This approach conflicts with our goal of transparent data transfer because the requestor has to tackle the data propagation explicitly.

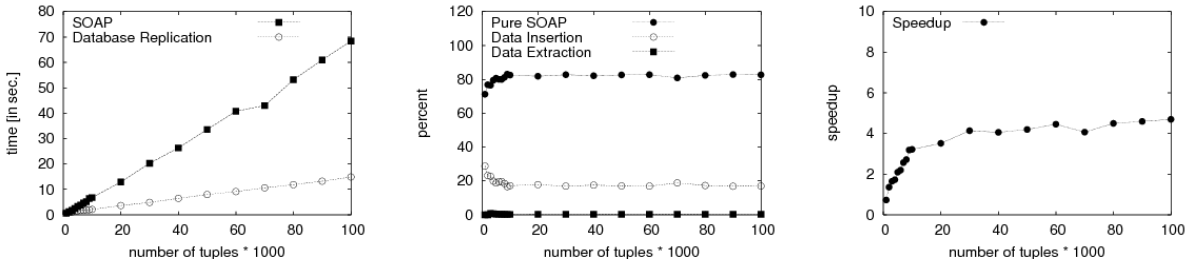
The second principle – *the pull approach* – stands in clear contrast to the push principle. Here, the Web service is responsible for the complete data transfer. The drawback of this principle is the monolithic approach, since all necessary functionalities regarding the data propagation must be available on the Web service side. This again conflicts with the following aims: (i) transparent data transfer between both partners (since the Web service has to tackle the data propagation) and (ii) seamless extensibility (because the data propagation tools must be integrated on the server side).

The most flexible approach is to use *mediators* as 3rd parties providing the data propagation tools behind Web service interfaces. These mediators are used to initialize the corresponding tools. The advantage of such mediators is that (i) the transparent data propagation is possible, (ii) Web services have to care for the pure functionality only and (iii) flexible and scalable integration of new data propagation infrastructures is possible.

In the first part of this section, we described our data-grey-box Web service interface extension; in particular, we introduced a new data binding format. To handle this new binding, we extend the SOAP framework by the integration of a novel data-layer component. The entire invocation process of our data-grey-box Web service is illustrated in Figure 8.

On the client side, enhanced Web service call semantics are necessary. Aside from the transmission of the endpoint and regular parameters in the SOAP message, the client has

Final edited form was published in "IEEE International Conference on Web Services (ICWS 2007), Salt Lake City, 2007". ISBN0-7695-2924-0.
https://doi.org/10.1109/ICWS.2007.69



(a) Times (b) Distribution (c) Speedup

Figure 9. Experiment 2 - SOAP versus Database Replication.

to deliver access information for (i) where the input data is available and (ii) where the output data should be stored. That means the new binding format is translated into no more than two additional parameters for access information for input and output data at the client site. These new parameters are included in the SOAP message for the invocation of Web services.

Actually, our data-grey-box Web services specify in the binding the type of the internally used data source. Furthermore, we include the types of data sources to be expected at the client site with the help of the additional attribute *required* in the binding, e.g. (required="database,xml-file"). Only requestors satisfying these constraints can invoke the Web service. For each possible data source type, our data layer includes an access specification. Based on these access specifications and the data binding, the structure of the additional parameters is created. The clients deliver the required access information according to the specification.

On the service side, our SOAP framework receives the SOAP message and conducts a separation into the functional aspect and the data aspect. The associated data layer can now call an appropriate mediator for the data propagation based on the access information of the client and the service. The data access information of the requestor is located in the received SOAP message, while the data access information for the service must be queried from our proposed *DCAS*, which is part of the infrastructure. Therefore, our DCAS is now involved in the invocation process of Web services. Either the appropriate mediator is known or it must be determined by a lookup in the UDDI registry.

Through the explicit integration of the DCAS in the WS invocation process, a data session is created on the service side in a first step. This session concept is used for the correlation of the data to the Web service instance. Furthermore, the WS instance knows its data session. As a second step, the mediator service for the input data is invoked synchronously. Once this data propagation step is complete, the input data is available on the service side and the proper functionality can be triggered. After the functional processing of the input data, a mediator service is invoked synchronously to transfer the output data to the service requestor. Once this transmission is complete, the WS invocation is finished.

An advantage of this invocation principle is that clients do not require to create a session explicitly, while a necessary data session is implicitly created during the Web service invocation. Therefore, our data-grey-box Web services are still stateless. Furthermore, the data propagation is transparent for both partners and new data exchange tools can be seamlessly integrated.

A drawback of the current solution is the requirement of static and typed access specifications in the binding. In the following step, we want to develop a dynamic solution. Furthermore, we are going to integrate a cost-based approach to decide which mediator should be used for the data propagation.

5. Evaluation

In this section, we evaluate our proposed approach regarding the performance gain. The foundation for all experiments is our ongoing example: A service requestor (realized as Web service) sends data from a local relational database system to our k-means Web service. This Web service processes the received data with the help of a relational database system. The application server JBoss 4 serves as implementation platform for our proposed approach. The service requestor and the Web service were running on different servers.

As separate data propagation tools, we used (1) the open-source ETL tool *Kettle* [11] and (2) a database replication approach. Those propagation tools ran on a third server behind service interfaces which are triggered during the Web service invocations. We measured the data exchange time from the service requestor to the Web service. The observation can be transcribed to the data transfer time from the Web service to the requestor.

The first conducted experiment was already described in Section 1. In this experiment, we transmitted $50,000$ tuples with different numbers of columns, where each column included string values of length 60. As described in Section 1, the data exchange with the ETL tool is sixteen times faster than the SOAP approach (see Figure 1(c)).

Final edited form was published in "IEEE International Conference on Web Services (ICWS 2007), Salt Lake City, 2007". ISBN0-7695-2924-0.
https://doi.org/10.1109/ICWS.2007.69

In the second experiment, we varied the number of tuples of the data set and we did not change the structure. The structure corresponds to a table with two double columns. Figure 9(a) shows the transmission times regarding different numbers of tuples for the SOAP and the data replication approach. As we can see, the database replication approach outperforms the SOAP approach. The larger the data, the larger the benefit (Figure 9(c)). Moreover, Figure 9(b) illustrates that the SOAP transfer, including XML serialization and deserialization at the requestor and the services, makes up almost 80% of the whole SOAP approach.

As the results of our experiments indicate, the integration of specialized data propagation tools is benefical with regard to the performance of the data exchange between service requestor and Web service.

6. Conclusion

In various application scenarios, like business-to-business communication or service-oriented computing, an transmission of large amount of structured data to and from Web Services is required. In service-oriented environments, the Simple Object Access Protocol has been established as the main transport protocol. However, the SOAP data transmission approach is not efficiently applicable indicated by our evaluation results. Furthermore, we have shown that spezialized data propagation tools outperforms the SOAP data transmission.

Therefore, we have proposed our data-grey-box Web services allowing the integration of data propagation tools in the Web service invocation procedure. In detail, we have discussed the following points: (1) We have introduced a data framework that is responsible for the handling of massive data sets within Web services; (2) Based on the first concept, we have enhanced the Web service interface with special properties regarding the data aspect; and (3) we have presented the invocation process for our data-grey-box Web services.

Our ongoing research activities cover two directions. One is that we want to further optimize our data-grey-box Web services in order to guarantee the optimized data propagation between service requestor and service. The other direction is concerned with the extension of WSBPEL by data transitions to allow the explicit consideration of data flows together with the function flow as well. Thereby, we expect extended optimization strategies on the one hand and modification options that can be controlled much more efficiently on the other hand so that we will be able to better react to dynamic changes.

References

[1] D. Andresen, D. Sexton, K. Devaram, and V. P. Ranganath. Lye: A high-performance caching soap implementation. In *Proc. of the 33rd International Conference on Parallel Processing*, 2004.

[2] M. Cai, S. Ghandeharizadeh, R. R. Schmidt, and S. Song. A comparison of alternative encoding mechanisms for web services. In *Database and Expert Systems Applications, 13th International Conference*, 2002.

[3] K. Chiu, M. Govindaraju, and R. Bramley. Investigating the limits of soap performance for scientific computing. In *11th IEEE International Symposium on High Performance Distributed Computing*, 2002.

[4] E. Forgy. Cluster analysis of multivariate data: Efficiency vs. interpretability of classifcations. *Biometrics*, 21:768, 1965.

[5] E. Gamma, R. Helm, and R. Johnson. *Design Patterns*. Addison-Wesley, 1995.

[6] M. Girardot and N. Sundaresan. Millau: an encoding format for efficient representation and exchange of xmlover the web,http://www9.org./w9cdrom/154/154.html.

[7] D. Habich, S. Richly, and W. Lehner. GignoMDA - Exploiting cross-layer optimization for complex database applications. In *Proc. of the 32nd International Conference on Very Large Data Bases*, 2006.

[8] S. Heinzl, M. Mathes, T. Friese, M. Smith, and B. Freisleben. Flex-swa: Flexible exchange of binary data based on soap messages with attachments. In *Proceedings of the IEEE International Conference on Web Services (ICWS'06)*, Washington, DC, USA, 2006. IEEE Computer Society.

[9] A. Hinneburg, W. Lehner, and D. Habich. Combi-operator: Database support for data mining applications. In *Proc. of 29th International Conference on Very Large Data Bases*, 2003.

[10] B. Kemme and G. Alonso. A new approach to developing and implementing eager database replication protocols. *ACM Trans. Database Syst.*, 25(3):333–379, 2000.

[11] Kettle ETL Tool. http://kettle.pentaho.org/.

[12] Mario Antoniolett et al. The design and implementation of grid database services in ogsa-dai. *Concurrency - Practice and Experience*, 17(2-4):357–376, 2005.

[13] A. Ng. Optimising web services performance with table driven xml. In *Proc. of the 17th Australian Software Engineering Conference*, 2006.

[14] L.-M. Patcas, J. Murphy, and G.-M. Muntean. Middleware support for data-flow distribution in web service composition. In *Proceedings of the combined Doctoral Symposium and 15th PhDOOS Workshop at the 19th European Conference on Object Oriented Programming(PhDOSS, Glasgow, Scotland, July 25)*, 2005.

[15] Service Data Objects. http://www-128.ibm.com/-developerworks/library/specification/ws-sdo/.

[16] R. van Engelen. Pushing the soap envelope with web services for scientific computing. In *Proc. of the International Conference on Web Services (ICWS'03)*, 2003.

[17] P. Vassiliadis, A. Simitsis, and S. Skiadopoulos. Conceptual modeling for etl processes. In *Proc. of the 5th ACM international workshop on Data Warehousing and OLAP*, pages 14–21, New York, NY, USA, 2002. ACM Press.

[18] P. Widener, G. Eisenhauer, and K. Schwan. Open metadata formats: Efficient xml-based communication for high performance computing. In *10th IEEE International Symposium on High Performance Distributed Computing*, 2001.