Request/Response Aspects for Web Services

Ernst Juhnke[1], Dominik Seiler[2], Ralph Ewerth[1],
Matthew Smith[3], and Bernd Freisleben[1]

[1] Department of Mathematics & Computer Science, University of Marburg
Hans-Meerwein-Str. 3, D-35032 Marburg, Germany
{ejuhnke,ewerth,freisleb}@informatik.uni-marburg.de
[2] Information Systems Institute, University of Siegen
Hölderlinstr. 3, D-57068 Siegen, Germany
d.seiler@fb5.uni-siegen.de
[3] RRZN, University of Hannover
Schloßwender Straße 5, D-30159 Hannover, Germany
smith@rvs.uni-hannover.de

Abstract. Web services rely on standardized interface descriptions and communication protocols to realize loosely-coupled distributed applications that are executed on several interconnected hosts. However, the extension of a web service with non-functional requirements, such as efficient data transfer or security, is a tedious task that also requires access to the web service implementations. In this paper, we present *request/response aspects* for web services to allow software developers to easily and transparently change the data exchange between web services *without* modifying their implementations or their interfaces. A framework supporting request/response aspects for web services is presented, and implementation issues are discussed. The usefulness of request/response aspects is illustrated by three use cases.

Keywords: Aspect-oriented Programming, Web Service, Service-oriented Architecture, SOAP.

1 Introduction

With the advent of service-oriented architectures (SOA) and web services as their most widely used implementation technology, applications can be composed of existing web services, promising higher reusability, faster development, and consequently, reduced costs. Web services are identified by their interfaces that in turn are defined using the Web Services Description Language (WSDL, [23]). A WSDL document contains a set of operations and defines input/output messages, faults and bindings for transport protocols. Typically, web services communicate via SOAP [22], relying on a request/response message exchange pattern based on XML documents. Given a set of web services, composition languages such as the Business Process Execution Language for Web Services (BPEL, [2]) can be used to compose them into a more complex service. The original services act as the basic activities in the newly constructed service; hence, this paradigm is

H. Mouratidis and C. Rolland (Eds.): CAiSE 2011, LNCS 6741, pp. 627–641, 2011.

often referred to as "Programming in the Large". The composite web service orchestrates the control flow between the original web services, stating the order in which their methods are called. To keep services decoupled, their interaction is managed by a client (i.e., a composer service like a workflow engine), and the services do not have any information about the service to be called next.

While powerful in terms of compositionality, the downside of the "Programming-in-the-Large" paradigm is that the control flow dictates its structure onto data-flow related concerns, which may not always be appropriate for them. We have experienced problems related to this dominance of the control flow structure during the development of a large-scale service-oriented platform for content-based search in image and video databases [10,11]. Following the service-oriented paradigm enables us to reuse multimedia algorithms, encapsulated as services, in different workflow configurations. However, it also forces data transfers between two subsequent multimedia services in a workflow to travel via the composition client in the middle, which is neither necessary nor desirable: The available network bandwidth of the machine hosting the BPEL engine can very quickly become a bottleneck and the runtime performance will decrease significantly.

One way to solve this issue is to avoid the transfer of huge data via the BPEL engine by using the Flex-SwA framework [12]. Instead of huge binary data, only a small reference is transferred from the data-producing service via the composition client to the data-consuming service. The data consuming service resolves the reference and gets the binary data. However, this technique requires that the involved services have reference data types in their method signatures and in their return values, respectively. Furthermore, modularity and decoupling of web services would suffer from scattered code for data handling.

It is preferable to have a solution that allows efficient data transmission in data-intensive service workflows without noticeable additional development efforts and without losing modularity and decoupling of services. One would like to be able to superimpose a structure over the service control flow such that the data transfer cuts across the control flow. Aspect-oriented programming (AOP) is a paradigm that is aimed at increasing the modularity of software [14]. Cross-cutting concerns such as the efficient transmission of data can be modularized in aspects. AOP enables a developer to integrate aspects into existing applications via join points, i.e., particular points in the control flow that specify when such modularized code (called "advice") should be executed; the description of a set of join points is called pointcut.

In this paper, we present *request/response aspects* for web services to address this problem. The proposed request/response aspects allow to add non-functional requirements at the web service communication level, and to execute aspect code on a remote host where a web service is running. They are independent of a web service's implementation language and do not assume that a web service implementation itself provides related functionality. For example, in the multimedia workflow mentioned above, the components for efficient data transmission can be woven dynamically into the communication infrastructure on top of which the web services run *without* changing their implementations or their interfaces.

We have implemented three use cases concerned with data transmission, data compression, and data encryption to demonstrate the feasibility of the proposed request/response aspects.

The paper is organized as follows. Section 2 reviews related work. The design of the proposed aspect framework for web services is presented in Section 3. In Section 4, implementation issues are discussed. The three use cases are presented in Section 5. Section 6 concludes the paper and outlines areas of future work.

2 Related Work

Several efforts have already been made to modularize cross-cutting concerns in a web service environment. Related approaches can roughly be grouped into three categories: (a) they operate on the composition of services; (b) they introduce an intermediary layer that encapsulates services; or (c) they work at the remote service, i.e. not a physically co-located service.

AO4BPEL [6] is an extension of BPEL to improve modularity and to support dynamic adaptation of the composition logic. In this setting, every BPEL activity is a possible candidate for a join point. AO4BPEL facilitates the modularization at the level of the BPEL engine. Despite the fact that due to its similarity to traditional programs this seems natural when weaving aspects into a service-oriented architecture, it is not possible to realize a data transfer aspect like the one described in our video analysis workflow, because an adaptation of the service is necessary.

Courbis and Finkelstein [8] present an adaptable BPEL infrastructure that is extensible both statically and dynamically. The extension is not limited to the engine itself, but also includes a BPEL process. While this is an interesting concept for developing a BPEL engine, this approach is – again – clearly limited to the adaptation of the BPEL engine and the BPEL process.

Other approaches [7,13] have also identified the necessity of modularizing crosscutting concerns in a service-oriented environment. To achieve this objective, they modularize scattered concerns at the composition level. This might be sufficient, e.g., for expressing authentication against remote services. However, the mentioned approaches cannot be used when crosscutting concerns affect the functionality of a remote service.

The Web Services Management Layer (WSML, [21]) introduces an intermediate layer between client and web service. The layer is used for the just-in-time integration of web services into a client. Thus, the scope of WSML is limited to the client-side. The thereby induced scattered code (by using multiple services) can be faced by utilizing Aspect Beans and Connectors of JAsCo [18]. JAsCO can work with client requests, but not with the requests on the service side.

Binder et al. [5] have introduced Service Invocation Triggers, a lightweight infrastructure based on a proxy layer that routes messages and thereby optimizes the data transfer when workflows are orchestrated. If the proxies are located on the same host as the service the proxy communicates with, a comparable situation to Flex-SwA is achieved in terms of optimizing the data transfer. However,

this is achieved at the expense of transferring the control flow to the proxy architecture that in this way represents a hidden orchestration layer, whereas in our approach it remains at the orchestration engine.

DJCutter [16] is a framework that provides remote pointcuts as a new language construct for distributed aspect-oriented programming. While an aspect is distributed to remote machines, it only notifies a central server when a pointcut becomes active. In turn, this server executes the corresponding advice. Referring to the data transfer example again, the data must first be transferred to the server that processes it and afterwards transferred back. This will certainly undo the desired performance gain.

Baligand and Monfort [4] present a framework to separate crosscutting concerns within the service implementation, but do not deal with crosscutting concerns over multiple services or hosts. Their focus is on the weaving of aspects into the (Java) byte code of the service implementation. Thus, they pin the presented framework to web services implemented in Java, whereas request/response aspects operate on the message level. This allows us to be independent of the concrete implementation language of a service.

AWED [15] is an aspect-oriented programming language with explicit support for the distribution of aspects and advice. In contrast to request/response aspects, it operates on Java. However, web services can be implemented in any programming language, and only their XML-based SOAP-interface is known. Again, it cannot be assumed that Java is the implementation language of web services, such that AWED cannot be used in web service environments where Java is not used as the implementation language.

3 Request/Response Aspects

In this section, the pointcut description for defining request/response aspects in web service environments and a supporting framework are presented. The pointcut description is independent of any implementation language, whereas the framework itself is located at the web service middleware (i.e., a software stack that provides support for SOAP communication, like Apache Axis [3]) and provides the capabilities of interpreting pointcuts and applying the corresponding advice. The framework consists of: 1.) an extension of the web service middleware that enables the weaving of aspects into web services without changing their implementation or their interface, including an aspect configurator service, a request/response aspect weaver, and a security manager; 2.) tools that operate at the client side, in particular for the seamless integration in BPEL workflows. This framework is an essential prerequisite for writing and weaving request/response aspects. By remaining independent of the service implementation, the framework respects the black-box idea of service-oriented architectures.

3.1 Framework for Request/Response Aspects

The execution of web services is typically performed by an installed middleware, such as a web service container and a SOAP processor. The underlying idea of

the proposed framework is to weave aspects into the SOAP message processing chain. The weaving of aspects at this point enables us to be independent of the concrete service implementation: the processing chain typically operates on XML documents and weaving into it does not interfere with the service implementation. In advance, an aspect configurator enhances the existing middleware by allowing to weave aspects and, e.g., to check whether it is allowed or possible to weave an aspect into a service based on the aspect definition and the enforced security constraints. The framework is based on the following design considerations:

Dynamic weaving: Since the implementation language of a web service is typically unknown, and the client's and the service's administrative domain differ, aspect weaving must be performed dynamically. As mentioned before, the middleware container hosting web services should expose web service methods for adding and removing aspects during runtime.

SOAP message chain: Due to the contract-first conception of web services, aspects are not allowed to change the previously negotiated interface (i.e., the WSDL description) of a service. Ideally, the aspects have to be placed in the SOAP processing chain to circumvent a modification of the interface. They must be able to deal with SOAP request and response messages such that a modification of them does not change their syntactical representation.

Caller/callee interaction: On the client side, the framework must have the ability to distinguish between different web service invocations and communication partners who are possibly affected by an aspect. An aspect affecting two subsequent services can only be added to a service if the corresponding aspect on the subsequent service can also be applied. On failure, the aspect has to be removed. Consequently, an atomic behavior of aspect weaving must be ensured. This includes the possibility for a caller to check whether an aspect is available that can be enabled, and to determine which aspects are currently active for this particular caller.

Scope: Aspects must have a certain scope [19]. A scope specifies whether the aspect is only valid for the client that has woven the aspect or whether the aspect is valid for all callers.

Security: A security manager must ensure that only authorized clients and aspects make use of the framework. The security manager can enforce that only aspects that meet security requirements can be used or deployed. Using established authentication mechanisms (e.g., SSL/TLS with client verification [20]), only authorized users are permitted to use the aspect configurator and to deploy signed aspects.

State: Since web services are stateless, aspects and advice should not maintain any internal state, besides the information stored in the external key value store (see 3.2) and in the message itself, respectively.

In Figure 1, the general design of the proposed framework for request/response aspects is shown. Our framework operates on the server side as well as on the client side. At the server side, there is a configuration interface (to be called by the client), a configuration manager and a component that performs the actual

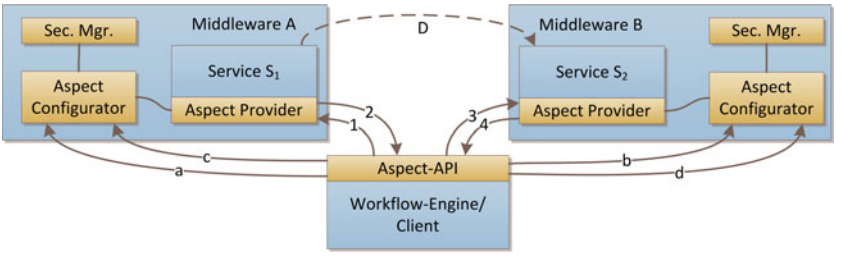

Fig. 1. Web service invocations with aspects

weaving of aspects. At the client side, mainly an API is provided that offers methods for interacting with the server component.

The interaction between a client and the framework is as follows. First, the client's Aspect-API asks the aspect configurators at the web services S_1 and S_2 whether they support the needed aspect (Figure 1: steps a and b). If both sites support the aspect and respond accordingly, then the aspect is added by the client (steps c and d) into the response of S_1 and the request of S_2. When the actual SOAP request is sent to Service S_1 (step 1), it is answered by the response message (step 2). Then, the client forwards the request (step 3) to Service S_2 that due to the request aspect can process the request and eventually responds (step 4). In our use case for data transfer, an aspect that supports direct and efficient data transmission from Service S_1 to Service S_2 will be used. Due to the use of request/response aspects, the response message of Service S_1 does not include the data D in this scenario, since the aspects enable the direct data transmission of data D to Service S_2.

3.2 Pointcut Description

A request/response aspect for web services is defined by the following tuple:

$$\mathcal{A} = \{\mathcal{P}, \mathcal{O}, \mathcal{F}, \mathcal{M}, \mathcal{I}, \mathcal{K}, \mathcal{D}, \mathcal{C}\}\,.$$

$\mathcal{P}$ (*porttype*) defines the qualified name (QName) of the web service porttype the aspect should be applied to. This element is required to cope with a wildcard operator, meaning that all porttypes within a service are relevant for this particular aspect. $\mathcal{O}$ (*operation*) specifies the name of the operation of the given porttype. A support for a wildcard operator is also needed. $\mathcal{F}$ (*field*) determines the actual WSDL message part the aspect should be applied to. Since XML schema elements can be nested complex data types, this field is realized as an XPath expression[24] , in case the aspect should be applied only to a particular part of a complex type. The XPath language also allows the use of wildcard operators and thus enables the application of the aspect to multiple elements or to multiple parts of (multiple) elements. $\mathcal{M}$ (*mode*) is an element of the enumeration containing the values {`request`, `response`, `both`}. The first one means that the aspect is applied to the request message, the same holds for `response`. When the mode is set to `both`, the aspect is applied to the request as well as to

the response message. $\mathcal{I}$ (*ID*) determines a reference to the actual advice that has to be applied by providing an ID containing a reference to the advice.

The preceding elements are mandatory, whereas the following elements are optional: $\mathcal{K}$ (*key value store*) provides configuration data in the form of key-value pairs optionally needed by the aspect. $\mathcal{D}$ (*depends*) and $\mathcal{C}$ (*conflicts*) contain a list of aspects that refer to woven aspects. The first one enforces the listed aspects to be woven, whereas the latter one forbids them to be applied. This enables the weaving operation to respect (in-)compatibilities between different aspects.

The pointcut description is represented by the XML schema type shown in Listing 1.1. The first six elements of the tuple, namely $\mathcal{P}$, $\mathcal{O}$, $\mathcal{F}$, $\mathcal{M}$, $\mathcal{I}$, $\mathcal{K}$, are mapped to the schema type. The fields $\mathcal{C}$ (conflicts) and $\mathcal{D}$ (depends) are not present in the schema type, because they are expressed by the implementation of the advice itself. Hence, they are omitted in the schema declaration.

```
<complexType name="Aspect">
 <sequence>
    <element name="portType" type="xsd:QName" />
    <element name="operationName" type="xsd:string" />
    <element name="field" type="xsd:string" />
    <element name="mode" type="xsd:string" />
    <element name="aspectPlugIn" type="xsd:string" />
    <element name="aspectData" type="tns1:HashMap" />
 </sequence>
</complexType>
```

Listing 1.1. XML Schema type of an aspect

4 Implementation

Our Java-based, prototypical implementation[1] of the aspect framework rests upon Apache Tomcat 6 as the application container in combination with Apache Axis 1.4 as the SOAP processing engine and web service execution environment. On the caller's side, the Axis client libraries are used. The orchestration engine for composing web services is ActiveBPEL [1], an open-source implementation of the BPEL standard.

Adding and removing of request/response aspects and listing of available advice are performed by a single web service that offers the corresponding methods. Further methods provided by this aspect configuration service are (1) a method to check whether an advice of an aspect is supported, and (2) a method to determine whether an aspect is woven into a service. These two methods allow the client-side component to ensure the atomic behavior of the overall framework.

[1] The source code is available on request.

4.1 Advice Interface

An advice of a request/response aspect is referenced by its ID (element $\mathcal{I}$ of the pointcut description). In our implementation, this ID represents the base name of the Java class that actually implements the advice. A class representing an advice has to implement a specific Java interface. Its methods are called by the aspect framework when a join point shadow becomes a join point (i.e, a defined join point is triggered) for the specific advice. The argument passed to these two methods is resolved by the expression in the field element $\mathcal{F}$ (see pointcut description) of the aspect. Their return values substitute the original value. If a wildcard operator is in the field element $\mathcal{F}$, multiple attributes match this expression. Multiple attributes represent multiple fields in a complex data type. The corresponding advice is applied iteratively to each of these attributes.

The concrete value of the scope of an aspect is either local or global with respect to communication. The value `local` means that the join point of the aspect respects the client, meaning that it is only active for the client that has woven it in, whereas `global` indicates that an aspect is active for all clients. The `global` scope implicates that no other service is affected by this aspect, because the affection depends on the actual client in order to determine the succeeding service. Since web services are loosely-coupled, such a succeeding service cannot be identified in general. Advice must furthermore define whether there is a subsequent service that is affected by this advice and to which an aspect has to be applied. Furthermore, this value (`none`, `direct`, `dataflow`, or `controlflow`) controls how such a service is detected. For example, an aspect for profiling the communication returns `none`, because there is no other partner. The value `direct` might be used for reliable messaging since it affects both partners that are communicating directly.

4.2 Aspect Configurator

The *aspect configurator* is responsible for validating and weaving aspects and is realized as a dedicated web service. To decouple the aspect framework from the conventional web services, the aspect configurator is implemented as a distinct service, i.e. a remote interface for weaving aspects. It operates on a registry that contains all necessary information about woven request/response aspects. Since it is realized as a hashtable, the number of deployed aspects has only a marginal impact on the overall aspect-weaving runtime.

4.3 Aspect Provider

The activation of (web service) join points is realized as an AspectJ aspect, the *Aspect Provider*. It is woven into the global handler chain of Apache Axis. If a web service is called or sends its response, the around advice is executed. It checks the registry to find out whether a request/response aspect has been woven into this concrete web service. If yes, the scope and the caller are identified, and in case of a complete match, the advice of the request/response aspect gets executed. For this purpose, the pointcut of the AspectJ aspect matches the

`invokeMethod()` method of the `RPCHandler` of Apache Axis, which in turn is in charge of invoking the concrete implementation of the web service. In Listing 1.2, the pointcut (line 1 – 5) and the advice (line 7 – 13) of the AspectProvider is shown. The AspectProvider is integrated into the handler chain of Apache Axis. Request/response aspects are applied in the order in which they were woven. A more sophisticated management strategy is subject to future work. The implementation as an AspectJ aspect circumvents the modification of the configuration of Axis (e.g., deploying a specific aspect deployment handler into each service) and potentially allows us to use this code within other middlewares supporting web services, such as JBoss or Spring.

```
pointcut invokeMethod( MessageContext msgC,Method method,
 Object obj,Object[] args) : call(
 protected Object RPCProvider.invokeMethod(
  MessageContext,Method,Object,Object[]) throws Exception)
  && args(msgContext, method, obj, argValues);

Object around(MessageContext msgC,Method method,Object obj,
 Object[] args) throws Exception :
 invokeMethod(msgC,method,obj,args) {
    handleRequest(argValues,method,msgC);
    Object response = (method.invoke(obj, args));
    return handleResponse(response,method,msgC);
 }
```

Listing 1.2. AspectProvider

4.4 Security Manager

The security manager mentioned in Section 3.1 uses a public key infrastructure based on the X.509[9] standard in order to authenticate users who call the aspect configurator. If an authorized user tries to deploy an aspect, the signature of the aspect is validated, and it is only deployed into the system if the validation is successful. Otherwise, the call fails and no aspect is deployed.

4.5 Aspect Invocation Handler

To facilitate the use of request/response aspects during service orchestration, a custom invocation handler of the ActiveBPEL engine is called each time a web service is called within a BPEL process. The weaving of aspects is initiated by the Aspect-InvokeHandler (AIH). The AIH includes all the client-side functionality described in Section 3 and is called every time the workflow engine performs a web service invocation. In order to register and use the AIH, an extension mechanism provided by the workflow engine itself is utilized. Thus, a modification of the implementation of the workflow engine is not necessary. When the AIH is called, it checks whether the actual web service should use an

aspect. This information is provided by the workflow developer during the design of the workflow. If this is the case, the AIH deploys the aspect to the actual web service and – depending on the defined relevance of the (request/response) advice – deploys it also to subsequent web services.

5 Evaluation

In this section, the realization of three use cases for request/response aspects is discussed. First, the use case of efficient data transmission described above is considered. The goal is to realize efficient data transmission between web services without additional implementation efforts (except for developing the aspects themselves once), without changing the web services, and without losing modularity and decoupling of web services. The second use case is also motivated by the multimedia workflow and realizes data compression. The third use case provides a cryptographic data transfer via request/response aspects. This use case is motivated by a workflow that performs medical analysis.

5.1 Use Case 1: Data Transfer

For the first use case, consider the general design shown in Figure 1 – before the request of Service S_2 can be handled by an aspect, the response of Service S_1 must have been handled by an aspect, too. In this way, the aspect woven into Service S_1 implements the `IResponseAdvice`. This implementation first takes the binary data returned by the service and then creates a new Flex-SwA reference that points to the data. This reference is encoded into the response to fit syntactically into the data structure the service returns. After this (response) message has been passed to Service S_2, the aspect woven into this service implements the `IRequestAdvice`. This implementation expects a Flex-SwA reference encoded in the data structure the service receives. Then, the encoded reference is resolved, and the data is transferred.

To test the Flex-SwA advice on a broader variety of data types, three different service implementations were investigated. The first echo service implementation works on a byte array, the second service on strings, and the third service uses SOAP with Attachments. The performance is evaluated using these three web services and the Flex-SwA advice mentioned before. This specific advice handles the request as well as the response message of the services and also deals with their (different) message formats. A client that utilizes the presented framework only needs to call the aspect configurator service with an aspect (see Listing 1.3) as an argument. This is all a client has to do in order to perform the weaving of the aspect for efficient data transmission.

Obviously, it can be expected that the weaving of the Flex-SwA aspect improves the workflow runtime significantly. The tests were performed on three machines running Fedora Linux. Each machine has the same hardware, namely an Intel Core 2 Duo E8600, 4 GB of RAM and a 100 MBit/s Ethernet network. One of these machines operated as the client, whereas the other two hosted the echo services. Different transmission types and different files sizes were used, but

not all tested transmission types support arbitrary data sizes. Due to the XML encoding – especially of arrays – the test of a byte array was limited by the available heap space for the Java virtual machine. The heap space was set to 2 GB and the Java Garbage Collector was allowed to run concurrently. Only with this setup we were able to test the transmission of byte arrays containing up to 800,000 single byte values.

```
Aspect serviceaAspect = new Aspect(
   new QName("http://fb12.de/AosStringTestService",
     "AosStringTestService"), "echoStringA", "/data",
   Aspect.AOP_RESPONSE_MODE, "FlexSwAPlugIn")
```

Listing 1.3. Java bean constructor of an aspect

We have compared the workflow execution times for the three different echo service scenarios (each measurement was repeated 100 times) to show their relative speedups. The less effective the data transfer mechanism is, the higher the (relative) runtime improvement is, if request/response aspects are used. In case of the byte array, a large improvement of up to 50% could be achieved, i.e., the transmission of a byte array in the size of 800 kbytes took about 95824 sec using plain SOAP communication and about 50190 sec using request/response aspects in combination with Flex-SwA. The runtime improvement of up to 50 % can be explained as follows: As indicated in Figure 1, each SOAP transmission first requires a serialization, then the actual transmission over a network and finally a deserialization – repeated in each step 1 – 4. Thus, the overall runtime time can be determined as $T_{\mathrm{SOAP}} = 4 * t(n) + \varepsilon$, where $t(n)$ is the network transmission time, including the serialization and deserialization time, and n is the amount of transmitted data (ε represents negligible processing times). In case of the Flex-SwA request/response aspect, step 2 and step 3 now transport the woven reference instead of the actual payload. Since the size of a reference is independent of the referenced amount of data, the runtime can be expressed as $T_{\mathrm{R/R}} = 2 * t(n) + 2 * t' + t_D(n) + \varepsilon$, where t' is the corresponding time for transferring a reference and $t_D(n)$ is the time needed to transfer the payload via Flex-SwA. For large n, we obtain $t' \ll t(n)$, $t_D(n) \leq t(n)$ and thus we get $T_{\mathrm{R/R}}^{n\to\infty} = 2 * t(n) + t_D(n)$. The comparison of T_{SOAP} and $T_{\mathrm{R/R}}^{n\to\infty}$ indicates that the theoretical runtime improvement is (slightly) below 50 %. Our measurements show that the proposed framework can come close to this theoretical limit. On the other hand, the overhead introduced by the request/response aspect is noticeable for the small data sizes when using strings or SOAP with Attachments (cf., Figure 2), but for larger data sizes ($>$ 40 KBytes) the achieved runtime improvement outweighs this overhead.

It is worth mentioning that not only the execution time of such a workflow can be accelerated by a significant factor, but also the development time is shortened considerably. This reduction in development time results from the fact that simple data types can be used for the development of the services and that

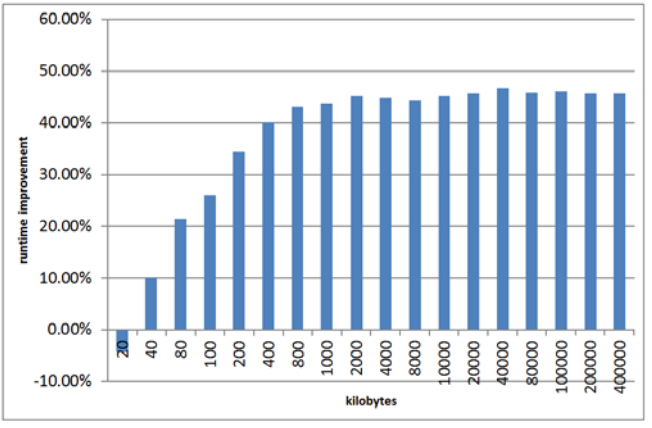

Fig. 2. SOAP with Attachments (SwA) – relative runtime improvement using Flex-SwA aspects

the more complex reference component for data transfer can be introduced using the proposed aspect framework. While the measurements only show the benefits for synthetical echo services, applying it to the concrete multimedia workflow also leads to a significant speedup. The plain workflow execution times without aspects are about 4422 seconds (where the SOAP communication requires 94% of the execution time). After weaving aspects into the services, the overall runtime only takes about 245 seconds (where now only 8% of this execution time is needed for communication). The overhead for the aspect handling (i.e., the time needed for executing the AspectProvider) is about 11 seconds.

5.2 Use Case 2: Data Compression

Text detection in videos is another workflow of our motivating multimedia analysis scenario: here, large text documents might be generated, depending on a given input video. These documents can be transported by Flex-SwA again, but it is also desirable to compress the text data. We implemented an advice to compress data and encode it afterwards with BASE64 in order to embed it again in an XML document. This aspect can reduce the message size to 60% of the original size. Another possibility would be to use the message-based compression offered by the web service container. Aspect-based compression allows us to use field-specific compression algorithms for different parts of the message, which is possible by the field operator of the pointcut description. This is reasonable since different types of data (text, images, videos etc.) may require different compression techniques. If a better compression algorithm becomes available, it is much easier to replace the applied compression using the proposed aspect-oriented solution. Otherwise, a new compression approach would have to be realized in all related web services' implementations.

This example indicates that the incorporation of new non-functional requirements via an advice is straightforward, since it effectively represents a filter

operation applied to basic data types. Such advice only need to be applied to the primitive data types to be employed by the proposed framework.

5.3 Use Case 3: Data Encryption

Secure messaging is another crosscutting concern that can also be handled by request/response aspects. The secure messaging problem is illustrated by a workflow that we have developed during a cooperation with medical researchers. When performing patient studies, it is important that personal data of patients is kept private by either making it anonymous or by encrypting it. For this reason, we have developed an advice that performs an encryption at the data source, such that all subsequent services are not able to read sensitive data. Eventually, the decryption aspect is located at the service that merges the results into a patient's record. By using such an aspect, services can be encrypted without changing the service implementation or configuring the middleware.

The workflow that motivates this use case originates from the area of sleep research and basically performs an ECG (electrocardiogram) analysis and, based on the obtained results, conducts apnoea detection. The implementation uses the Physio Toolkit [17], a common set of open source tools in the biomedical sciences. Since the data format of the recorded vital signs is different from the format required by the Physio Toolkit, a data conversion is needed. Afterwards, the ECG records are processed to detect medically relevant peaks in the signal. The results are passed to an annotation reader service that in turn decodes the input and passes the results to a beat detection service that detects particular waves within the signal. In parallel, the output is passed to the apnoea detection service that analyzes the signal and detects respiration dropouts to diagnose the sleep apnoea syndrome.

The data exchanged by the services contain the actual ECG measurements and also some identification attributes. To prevent the misuse of these attributes, we have developed a privacy advice that uses public key cryptography. The support of the wildcard operator allows us to encrypt (and decrypt) all of the patient related data. These data are encrypted when they are initially retrieved from a database. During the processing by the mentioned services, the personal information is encrypted (while the ECG data remain unencrypted). Finally, when the analysis is finished and the result is stored in the database, the corresponding advice decrypts the personal data.

6 Conclusions

In this paper, we have proposed request/response aspects for web services that allow developers of service-oriented applications to easily enrich web services with additional non-functional requirements, such as efficient data transmission, data compression, or other crosscutting concerns. They can be woven dynamically into remote web services without changing their implementations or their interfaces. The presented framework supporting request/response aspects includes a pointcut description for SOAP-based web service environments.

Request/response aspects offer several advantages: They allow adding non-functional requirements at the web service communication level to offer the possibility of executing aspect code on the remote host where a called web service is running. Furthermore, they are independent of a web service's implementation language and do not assume that the web service implementation provides related functionality. By using the aspect framework, the development of web services is simplified. This is demonstrated by adding the non-functional requirement of efficiently transmitting large amounts of data in a web service workflow and thus circumventing the bottleneck at the client or workflow engine, respectively. Runtime measurements for a multimedia application that requires efficient transmission of large amounts of data have been presented. Two further use cases for data compression and encryption have demonstrated the benefits of the proposed approach.

There are several areas for future work. For example, instead of transferring the aspect-ID ($\mathcal{I}$), the whole aspect could be copied either as (Java) binary code or as an interpretable description to the remote service. Ideas like sequence pointcuts (sophisticated management of multiple advice for the same pointcut) and shared states between aspects executed on different hosts [15] are other interesting enhancements of our approach. Finally, to prevent a congestion of a service with aspects over time, investigations for sophisticated life cycle management (e.g., according to wall-clock time or communication patterns) are areas of further research.

Acknowledgements

This work is supported by the German Ministry of Education and Research (BMBF, D-Grid) and by the German Research Foundation (DFG, PAK 509).

References

1. ActiveEndpoints: ActiveBPEL Business Process Execution Engine, `http://www.activebpel.org`
2. Andrews, T., Curbera, F., Dholakia, H., Goland, Y., Klein, J., Leymann, F., Liu, K., Roller, D., Smith, D., Thatte, S., Trickovic, I., Weerawarana, S.: Business Process Execution Language for Web Services Version 1.1. Microsoft, IBM, Siebel, BEA und SAP, 1.1 edn. (May 2003)
3. Apache Foundation: Apache Axis., `http://ws.apache.org/axis/`
4. Baligand, F., Monfort, V.: A Concrete Solution for Web Services Adaptability Using Policies and Aspects. In: Proc. of the 2nd Intl. Conf. on Service Oriented Computing, pp. 134–142. ACM, New York (2004)
5. Binder, W., Constantinescu, I., Faltings, B.: Service Invocation Triggers: A Lightweight Routing Infrastructure for Decentralized Workflow Orchestration. In: Intl. Conf. on Advanced Information Networking and Applications, vol. 2, pp. 917–921 (2006)
6. Charfi, A., Mezini, M.: Aspect-oriented Web Service Composition with AO4BPEL. In: Proc. of the European Conf. on Web Services, pp. 168–182. Springer, Heidelberg (2004)

7. Cibrán, M., Verheecke, B.: Dynamic Business Rules for Web Service Composition. In: 2nd Dynamic Aspects Workshop (DAW 2005), pp. 13–18 (2005)
8. Courbis, C., Finkelstein, A.: Towards an Aspect Weaving BPEL Engine. In: The Third AOSD Workshop on Aspects, Components, and Patterns for Infrastructure Software (ACP4IS), Lancaster, UK, pp. 1–5 (2004)
9. Cooper, D., Farrell, S., Boeyen, S., Housley, R., Polk, W.: Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile, http://tools.ietf.org/html/rfc5280
10. Ewerth, R., Freisleben, B.: Semi-Supervised Learning for Semantic Video Retrieval. In: Proc. of the 6th ACM Intl. Conf. on Image and Video Retrieval, pp. 154–161. ACM, New York (2007)
11. Ewerth, R., Mühling, M., Freisleben, B.: Self-Supervised Learning of Face Appearances in TV Casts and Movies. In: Proc. of the Eighth IEEE Intl. Symposium on Multimedia, pp. 78–85. IEEE Computer Society, Los Alamitos (2006)
12. Heinzl, S., Mathes, M., Friese, T., Smith, M., Freisleben, B.: Flex-SwA: Flexible Exchange of Binary Data Based on SOAP Messages with Attachments (2006)
13. Joncheere, N., Deridder, D., Straeten, R., Jonckers, V.: A Framework for Advanced Modularization and Data Flow in Workflow Systems. In: Proc. of the 6th Intl. Conf. on Service-Oriented Computing, pp. 592–598. Springer, Heidelberg (2008)
14. Kiczales, G., Hilsdale, E., Hugunin, J., Kersten, M., Palm, J., Griswold, W.: An Overview of AspectJ. In: Proc. of the 15th European Conf. on Object-Oriented Programming, pp. 327–353 (2001)
15. Navarro, L., Südholt, M., Vanderperren, W., De Fraine, B., Suvée, D.: Explicitly Distributed AOP using AWED. In: Proc. of the 5th Intl. Conf. on Aspect-Oriented Software Development, pp. 51–62. ACM, New York (2006)
16. Nishizawa, M., Chiba, S., Tatsubori, M.: Remote Pointcut: A Language Construct for Distributed AOP. In: Proc. of the 3rd Intl. Conf. on Aspect-Oriented Software Development, pp. 7–15. ACM, New York (2004)
17. PhysioNet: PhysioToolkit, http://www.physionet.org/physiotools/
18. Suvee, D., Vanderperren, W., Jonckers, V.: JAsCo: An Aspect-Oriented Approach Tailored for Component Based Software Development. In: Proc. of the 2nd Intl. Conf. on Aspect-Oriented Software Development, pp. 21–29. ACM, New York (2003)
19. Tanter, É.: Expressive Scoping of Dynamically-Deployed Aspects. In: Proc. of the 7th Intl. Conf. on Aspect-Oriented Software Development, pp. 168–179. ACM, New York (2008)
20. Transport Layer Security, http://datatracker.ietf.org/wg/tls/charter/
21. Verheecke, B., Cibran, M., Vanderperren, W., Suvee, D., Jonckers, V.: AOP for Dynamic Configuration and Management of Web Services. Intl. Journal of Web Services Research 1(3), 25–41 (2004)
22. World Wide Web Consortium (W3C): W3C SOAP Specification, http://www.w3.org/TR/soap/
23. World Wide Web Consortium (W3C): Web Services Definition Language (WSDL) 1.1, http://www.w3.org/TR/wsdl
24. World Wide Web Consortium (W3C): XML Path Language (XPath), Version 1.0, http://www.w3.org/TR/xpath, http://www.w3.org/TR/xpath