The Web Service Browser: Automatic Client Generation and Efficient Data Transfer for Web Services

Steffen Heinzl[1], Markus Mathes[1], Thilo Stadelmann[1], Dominik Seiler[1],
Marcel Diegelmann[2], Helmut Dohmann[2], Bernd Freisleben[1]

[1]Dept. of Mathematics and Computer Science, University of Marburg
Hans-Meerwein-Str. 3, D-35032 Marburg, Germany
{heinzl,mathes,stadelmann,seiler,freisleb}@informatik.uni-marburg.de

[2]Dept. of Applied Computer Science, University of Applied Sciences Fulda
Marquardstr. 35, D-36039 Fulda, Germany
{marcel.diegelmann,helmut.dohmann}@informatik.hs-fulda.de

Abstract

Web services are supported by almost all major software vendors, but nevertheless there is still a certain barrier that prevents a broader user community to actually use them. The barrier is the lack of appropriate clients offered in conjunction with the services. This paper presents a Web Service Browser that automatically generates a dynamic user interface when the user browses to the location of the service description and additionally handles the invocation of the service. To ease the use of the service, the browser takes care of data management by using an implementation of the Flex-SwA architecture. Results are presented to the user in a human-readable manner. When the result contains multimedia data, an audio or video player is used to present the result. Use cases demonstrate the benefits of the browser. With the Web Service Browser, web services simply become a usable component offered in the WWW.

1 Introduction

Although web services are normally used for machine-to-machine communication, there are areas like high-performance computing, e-government or multimedia services where humans communicate or interact with web services. However, there are still some obstacles for people who want to use web services. Most WSDL files found in the WWW miss a working and easy to use client for invoking the corresponding web services, and potential web service users have to write their own client software to test and use a web service. Even if clients are available, these are often built for specific platforms, such as the Microsoft .NET framework. This limits their use to people using the Microsoft Windows operating system or to computer experts who know how to compile or build their own client software. There are already applications that generate user interfaces (often as part of an integrated software development environment), but to really allow web services to target a broader user community, a familiar environment like a web browser that can be installed easily and that assists the user in the invocation of the service is desirable.

In this paper, a Web Service Browser is presented. When a user browses to the WSDL location of a web service, an intuitive user interface is generated automatically. Using the Web Service Browser, web services can be invoked easily. Data transfers are effectively managed to ease the sending of large amounts of data. Documentation tags in the WSDL/XML Schema allow to offer help to the user when choosing the operation or providing the parameters for the web service. If the service returns a result, it is presented according to the contained data. Textual and multimedia data are directly visualized or made audible, respectively.

The developed Web Service Browser also supports web services built according to the Web Service Resource Framework (WSRF), commonly called Grid services. The particular problems associated with handling Grid services have been published in a previous paper [3]. For Grid services, the `document/literal wrapped` binding style is supported. The Web Service Browser, however, supports the SOAP over HTTP binding and the different `style`/`use` combinations.

The paper is organized as follows. Section 2 describes user requirements with respect to web service invocation. Section 3 presents the components of the Web Service

978-0-7695-3709-2/09 $25.00 © 2009 IEEE
DOI 10.1109/ICWS.2009.22

Browser. In Section 4, parts of the implementation are discussed. Section 5 presents two use cases. Section 6 discusses related work. Section 7 concludes the paper and outlines areas for future research.

2 User Requirements

Ordinary computer users have different requirements with respect to web service invocation. If a user browses to the service's location with a web browser, (s)he normally sees an XML tree showing the WSDL file, as shown in Figure 1. Most users cannot build a working client from such a WSDL description.

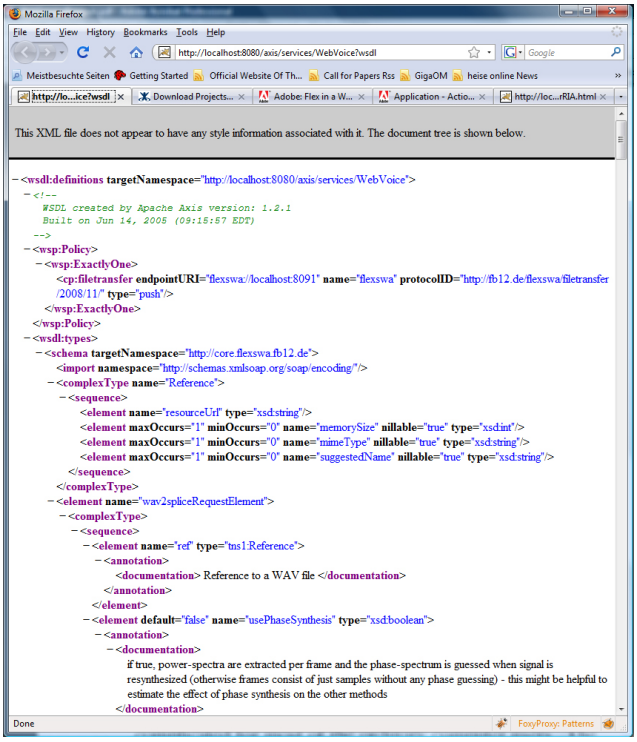

Figure 1. Screenshot of Firefox showing a WSDL file.

Therefore, the primary requirement for ordinary users is ease of use. The process of invoking a web service is too complicated for most of the ordinary computer users. The invocation of the first web service usually includes the following steps:

- installing a compiler for a programming language
- installing a SOAP engine that allows the creation of clients using the chosen programming language
- downloading the WSDL description
- creating stubs from the WSDL description
- implementing a client with the help of the stubs
- building and starting the client

Instead, it would be desirable that users simply access a WSDL file with their browser (like browsing to a HTML file) and then see a user interface asking for the information necessary to invoke the service. A simple operation (like clicking a button) should start the invocation of the service. When a result is returned, it should be shown to the user in a human-readable manner.

A second requirement is efficient data transfers, especially of large amounts of data. Normally, data is embedded into the SOAP message. This leads to a serialization effort, larger messages due to encoding, and a high memory usage since most SOAP engines manage the complete SOAP message (or at least a whole element) in memory before sending or processing it. If the messages get larger, memory problems may occur on the client machine. Thus, a framework managing data transfers in a flexible way is needed to circumvent the performance and memory problems.

A third requirement is to assist a user when entering complex data types, or when choosing the operation to invoke.

These requirements can be met by integrating the service handling procedure into a web browser. The user works in a familiar environment; services simply become a usable component offered in the WWW.

3 Components of the Web Service Browser

Conceptually, the Web Service Browser needs several components to assist the user in interacting with web services. First, a component is needed that parses and interprets the WSDL description and then fills a model holding the information contained in the WSDL. A second component has to generate a user interface (UI) based on the model. After the user has filled in the fields, a third component needs to generate the SOAP message according to the style/use combination defined in the binding. When a response is returned, a fourth component has to interpret the response and create an appropriate presentation of the result. Figure 2 shows the detailed sequence of actions, from retrieving the WSDL description over submitting the data to displaying the results.

After the user browses to a WSDL file, the following steps happen in the browser:

- The browser retrieves the WSDL description by sending a HTTP GET to the webpage or a SOAP message to a UDDI registry (1).

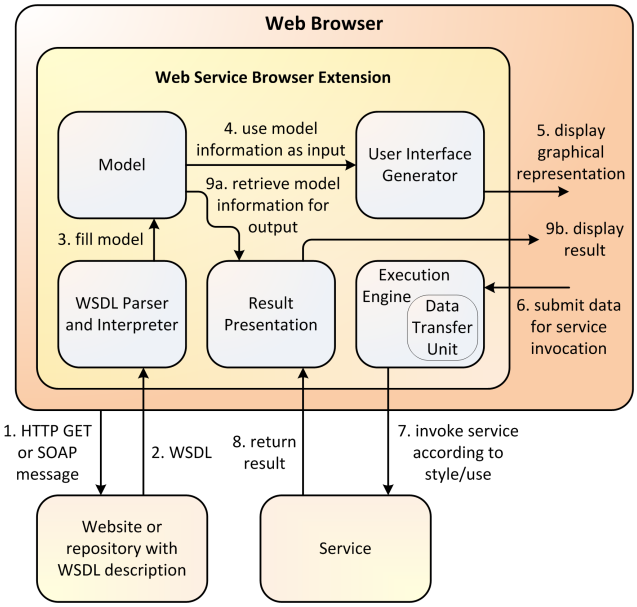

Figure 2. Invoking a service from the Web Service Browser.

- Instead of showing an XML tree (which is normally done by most browsers), the *WSDL Parser* parses and interprets the WSDL description (2).
- The data of the parsing process is used to fill a model holding the information of the WSDL file (3).
- The model information is then used as an input for a *User Interface Generator* (4) that creates a graphical representation.
- The representation is handed over to the browser (5).
- The user provides the data to an *Execution Engine* (6).
- The *Execution Engine* invokes the services according to the style/use combination provided in the service's binding (7).
- The service returns a result that is handed over to the *Result Presentation* component (8).
- The *Result Presentation* component uses the model information for the output structure of the result (9a) and displays the result in that structure (9b).

In the following sections, the different components are described in more detail.

3.1 WSDL Parser and Model

The *WSDL Parser* fills the *Model* with the information of the WSDL description including the XML Schema information of the WSDL types section. This information should be easily retrievable from the model. Furthermore, annotations and documentation should be accessible to later assist the user in choosing the correct operation and entering the correct data.

3.2 User Interface Generator

The model is then used as the input for the *User Interface Generator*. The User Interface Generator retrieves the information in the model to generate a graphical representation that can be implemented by making use of various styles and technologies. HTML forms, for example, provide a natural look to the user. Silverlight, Flash, Flex, JavaScript, etc. may provide a richer user interface, but a server-side component may be needed. Alternatively, the UI may also be presented as part of the browser (as, for example, in Firefox chrome [5]). Independent of the generated UI, the user must be able to enter the data needed for the service invocation.

From the `types` section of the WSDL description, different fields can be created. By using the XML Schema `appInfo` element or the `documentation` tag of XML Schema or WSDL, information can be provided about which graphical element to use for which field and which texts to display to help the user fill out the forms. For example, when XML Schema's `datetime` type is used, a graphical calendar to select the date can be used instead of having to enter a string in a hardly human-readable format.

To support file transfers, a framework for efficiently handling data transfers should be integrated into the Web Service Browser. An implementation of the Flex-SwA architecture (an overview is given in Section 3.4) fits nicely into the Web Service Browser's architecture. When a service uses the *Reference* type from Flex-SwA, the UI Generator should generate a field allowing a user to select a file from hard disk or to enter a URL referencing a file on a network or in the Web. Flex-SwA's data transfer capabilities are then used to transfer the file.

Using a User Interface Generator to automatically generate a UI has several advantages: A service provider does not have to offer a client or web interface for its services, and the service developer does not have to program a client for a web service. Furthermore, an efficient file transfer facility can be added to the UI. Even a non-computer expert can then easily transfer files and does not need to learn "new" techniques to transfer files (like SSH, GridFTP, etc.).

3.3 Execution Engine

The *Execution Engine* is responsible for actually invoking the service. It takes the data of the user and creates the SOAP message according to the

style/use combination defined by the service. The following style/use combinations exist and must be supported: `RPC/encoded`, `RPC/literal`, `document/encoded`, `document/literal`, `document/literal wrapped` (not being a real combination of style/use, instead depending on the schema).

After the creation of the SOAP message, the Execution Engine sends the message to the service. For the efficient transfer of bulk data, an extra Data Transfer Unit is used.

3.4 Data Transfer Unit

The *Data Transfer Unit* is used for efficient and flexible bulk data transfer. Since SOAP is inefficient for this task due to the required encoding effort, the Data Transfer Unit is based on the Flex-SwA architecture [4].

The Flex-SwA architecture provides a flexible way to handle bulk data in service-oriented environments. A *reference builder* is used to create an XML description that refers to the actual location of a file and the protocols used to transmit the binary data objects. The service provider uses a reference to retrieve the data directly from a remote server (on a client computer or anywhere else in the web) instead of transferring data from a remote server to the client and from the client to the service provider. A reference may use a MIME type to describe the referenced data. References do not need to be handled by a service provider directly but can be forwarded to other service providers with negligible additional communication cost. From an application developer's point of view, service invocation and data transmission remain coupled in a single service invocation operation. A service developer can choose between different communication patterns, thus configuring how the Flex-SwA platform should handle the referenced data, e.g. should the data be completely transferred before the service is executed or should the transmission overlap the service execution.

As an additional benefit, service developers can use the protocol handling capabilities of Flex-SwA to leverage high performance binary protocols by simply specifying a policy to use them, without having to deal with the protocol details in the application code. Binary protocols can be selected for each message part individually. Services may use a *communication policy* [2] to describe which protocols they support and a *temporal policy* [7] to add a validity period to the policies used.

For applications that repetitively send data to a service, the Flex-SwA architecture has been extended to enable references to point to data in memory identified by a unique ID [9].

3.5 Result Presentation

The *Result Presentation* component is responsible for creating the visualization of a result message that is then shown in the browser. The visualization might probably be similar to the way the input parameters are visualized. The user should be able to easily see what kind of data has been visualized. Besides textual data, a download link should be provided for bulk data. Furthermore, the Result Presentation should be extensible, for example, for multimedia data, such that an audio player is embedded for music, and a video player for movies.

4 Implementation Issues

The components of the Web Services Browser have been implemented as part of a Firefox 3 plugin. The plugin needs a Java version between 1.6.0_04 and 1.6.0_07. Newer Java versions that enable a preview to the new Java 7 features cause problems with LiveConnect that is used to connect Firefox with Java. The LiveConnect issues are presumably resolved in Java 1.6.0_12. The embedding of the audio player used in one of the use cases described later has been tested with Quicktime 7.6.

This section gives a short overview of the implementation of the plugin. The WSDL Parser component, the structure of the created HTML page, and the Result Presentation component are presented in more detail. The way the plugin integrates into Firefox (via XPCOM components and browser overlay) and the user interface generator and execution engine are explained in detail in a previous publication dealing with Grid service support [3].

4.1 Firefox Plugin

The Web Service Browser can be realized by extending a wide-spread browser such as Microsoft Internet Explorer or Mozilla Firefox. Both web browsers offer extension points to add functionality to them. Since the Internet Explorer can only be used on Windows operating systems, an extension for Mozilla Firefox was created. To avoid a reimplementation of a web service engine, it is reasonable to use an existing SOAP engine in conjunction with the Firefox plugin. For the Web Service Browser, Apache Axis was used. Two popular extension mechanisms were used to extend Firefox' functionality: overlays and Cross Platform Component Object Model (XPCOM) components.

Two XPCOM components have been developed. The first one is an observer, the second one a stream converter. The observer subscribes to three topics: `xpcom-startup`, `http-on-examine-response`, `http-on-modify-request`. The `xpcom-startup` subscription is persisted by adding a category entry to

the category manager during the registration process of the add-on. The category entry registers a persistent subscription to the other two topics. Now, whenever a HTTP response arrives, the observer reacts to an `http-on-examine-response` event and changes the content type from `text/xml` to `text/mywsdl`; `text/mywsdl` is a custom content type for which the stream converter component is registered.

The stream converter processes the HTTP response. If the document is a WSDL document, user interface generation will take place. The document will be replaced by a HTML page that contains HTML forms for the corresponding fields in the WSDL description. The generation of the HTML page is done in Java by using a Java Bridge—an adapted XPCOM component of the SIMILE project (`http://simile.mit.edu`). A browser overlay is used to initialize the Java Bridge, directly after the browser's main window has opened.

The generator returns a string containing the HTML page with JavaScript and some HTML forms. A Flex-SwA *Reference* type in the WSDL indicates a bulk data transfer. For each *Reference*, a file input that allows the selection of the file to transfer is generated.

JavaScript allows the user to select which port and port type to use, which operation to invoke, and assists the user to select which type to use for a message part. All inputs are collected when the user clicks the invoke button, and put into a SOAP message that is transmitted via an `XMLHttpRequest` to an Execution Engine. The `http-on-modify-request` topic allows the observer to cancel the HTTP request before Firefox submits it to the remote site. Instead, the Execution Engine is started locally via the Java Bridge.

The result is returned to the browser by identifying the tab that made the `XMLHttpRequest` and change its content. To achieve this, to each already opened tab an attribute named `gridbrowser-tab` is assigned with a consecutive number as its value. A `TabOpen` event listener is used to add the attribute to each newly created tab. This guarantees that each tab has an associated number that cannot be altered by moving or closing existing tabs or opening new tabs.

Right before the generation of the client, the stream converter retrieves the current tab by requesting a reference to the window mediator's most recent window. The associated `gridbrowser-tab` number is added to the `XMLHttpRequest`'s request headers of the generated code. When the user clicks the invoke button, the `XMLHttpRequest` possesses a request header with a reference to the tab that sent it. This tab number can then be obtained by the observer component from the HTTP request when it reacts to the `http-on-modify-request`. When the Execution Engine returns the result of the service invocation, this result can then be shown in the `result` section of the referenced tab.

4.2 WSDL Parser

The *WSDL Parser* used in the plugin is based on our implementation of an XML2Java Model Generator. The XML2Java Model Generator creates a Java class for each XML element in an XML file. For each child element, a corresponding object is added to the class. If multiple elements of the same type occur, then a list of the objects is added to the class. For each attribute, a string is added to the class with the corresponding getters and setters. For each list type, an `add()`-method is added to the class. Each created Java class furthermore contains a field `elementNamespace` to save the current namespace of the XML element and a field `elementValue` to save the string data between start and end tag. XML namespaces are converted to Java packages whereupon numbers (for example years) are preceded by an 'n', as packages may not start with a number.

The XML2Java Model Generator builds a "best practice" model. The model becomes finer the more input data it gets, i.e. it reflects the union of the structures of all XML example files. For policies that usually have a simple model, a short number of examples is sufficient to build a complete model of the policy in Java. For a complex specification like WSDL, more files are needed. An advantage of this approach is that the model is only as complex as the files to be processed. In the case of WSDL, there are many parts of the specification that might not be used, so the model does not reflect these parts. For example, the `definitions` tag in all the files encountered so far only has one `service` object, whereas according to WSDL 1.1 it could contain many services. In practice, a WSDL file is only used to describe exactly one service. By only taking into account the features that are really used, the complexity of the model is lower and hence it is easier to work with the model and, for example, build a parser on top of the model.

If the model is not sufficient for a specific WSDL file, it can simply be extended by being fed with the file. This approach is useful if the developer has in-depth knowledge of the XML specification. The Java representation reflects the structure of the XML files very closely and can thus be easily used.

4.3 HTML Structure

For the graphical presentation, we have chosen to generate an HTML page that completely abstains from using server-side components. Therefore, the necessary parts of the model information have to be reflected in the automatically generated HTML page. The creation of the SOAP

```
<div id="divop0.0" ...">...
  <span id="0:ref" class="http://core.flexswa.fb12.de"></span>
  <span id="0:ref:resourceUrl" class="http://core.flexswa.fb12.de">
    <input id="input1" type="file">
  </span>
  <span id="0:preservedBlockSize" class="...">
    <img src="..." onmouseover="showTooltip(...)" onmouseout="hideTooltip()">
    <input onmouseover="showInputTooltip(...)" onmouseout="hideTooltip()" value="12" type="text">
  </span>
  <span id="0:intermediateFrameCount" class="...">
    <img src="..." onmouseover="showTooltip(...)" onmouseout="hideTooltip()">
    <input onmouseover="showInputTooltip(...)" onmouseout="hideTooltip()" value="5" type="text">
  </span> ...
</div>
```

Figure 3. Schematic representation of the `wav2splice` operation.

body has been completely reimplemented in JavaScript and embedded into the HTML page, such that it can be used directly from the generated user interface. For each operation in each port type, a `div` container is added to the HTML page holding a `span` for each variable. The `id` attribute of a `span` describes which of the data belongs together, as can be seen in Figure 3 showing the schematic HTML representation `wav2splice` operation of the `WebVoice` Web Service (see Section 5.2).

The first `span` defines an element named `ref`. The `class` attribute is used to describe the namespace of the `ref` element, namely `http://core.flexswa.fb12.de`. The next `span` defines an element `ref:resourceUrl` that indicates that the `resourceUrl` element is part of the `ref` element. The other elements (`preservedBlockSize`, etc.) are top-level elements just like the `ref` element.

Dependent on the binding style, the SOAP message is created from the field information the user entered.

4.4 Result Presentation Engine

For the result presentation, different plugins can be implemented. Currently, the Result Presentation Engine visualizes textual information, the original SOAP message, and takes care of multimedia data. Whenever bulk data is part of the service's result, the service can use Flex-SwA References to reference the bulk data and assign a MIME type to them. Depending on the MIME type, different result presentations are used. For the MIME types `audio/x-wav` and `audio/mpeg`, an audio plugin has been implemented as part of the extension that embeds a player for and a download link to the referenced audio files into a new section of the HTML page. For the MIME types `image/png` and `image/jpeg`, a new section with previews of the pictures is created.

5 Use Cases

In this section, two different example web services are presented to show the feasibility of the Web Service Browser. The first service is a hotel booking service, and the second one is an audio analysis service.

5.1 Hotel Booking Service

First, we selected a hotel booking service as a typical example for a web service. It allows users to book a hotel by specifying their given name, surname, age, and e-mail address. The e-mail address is used for verification purposes.

The user simply browses to the WSDL file of the service. Instead of seeing an XML tree (as without the plugin), the user sees a user interface that allows him or her to add and submit data to the service. The screenshot in Figure 4 shows how the Web Service Browser automatically assists the user when selecting a date. Otherwise, the user would have to enter the XML Schema `datetime` type.

5.2 Audio Analysis Service

To show that the browser is extensible with more complex plugins and supports services with other than textual data types, a service from the area of automatic speaker recognition for multimedia analysis is presented as a second use case. It offers possibilities to "hear" what is going on inside different stages of the internal pattern recognition chain (i.e. different features and statistical models) by making intermediate results audible again. This eases the understanding and debugging of the underlying methods and results. Figure 5 shows the user interface of the audio analysis service called `WebVoice` that is generated when the user browses to the location of the service's WSDL description. Among others, the service offers the `wav2splice` operation that segments the signal into blocks of specifiable size

Figure 4. Screenshot of the hotel booking service user interface.

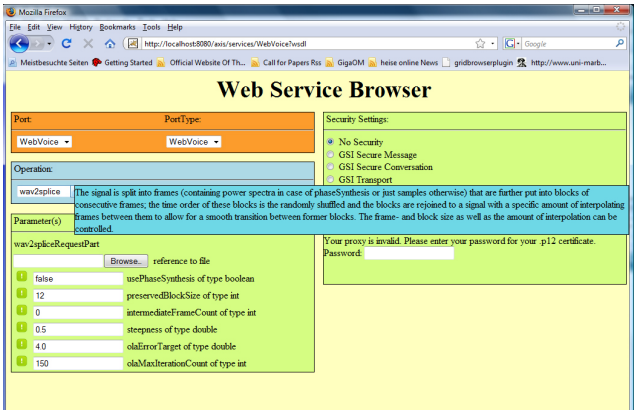

Figure 5. Screenshot of the `WebVoice` service user interface.

and randomizes the order of these blocks, helping to judge the importance of sequential information in the signal.

When the user slides over the operations drop down menu, a tooltip helps choosing the right operation. The form fields are already filled with the default values set in the XML schema part of the WSDL file. To transfer an audio file, the user pushes the browse button and selects a file from the hard disk. The audio file is then transferred to the web service via Flex-SwA. The data is pushed to the service when the user is in a private network. If the user has a public IP address, the local file is streamed to the service, such that the processing of the data can overlap with the transmission. After the analysis, the result presentation engine generates an HTML page offering the user to download or listen to the resynthesized audio (see Figure 6).

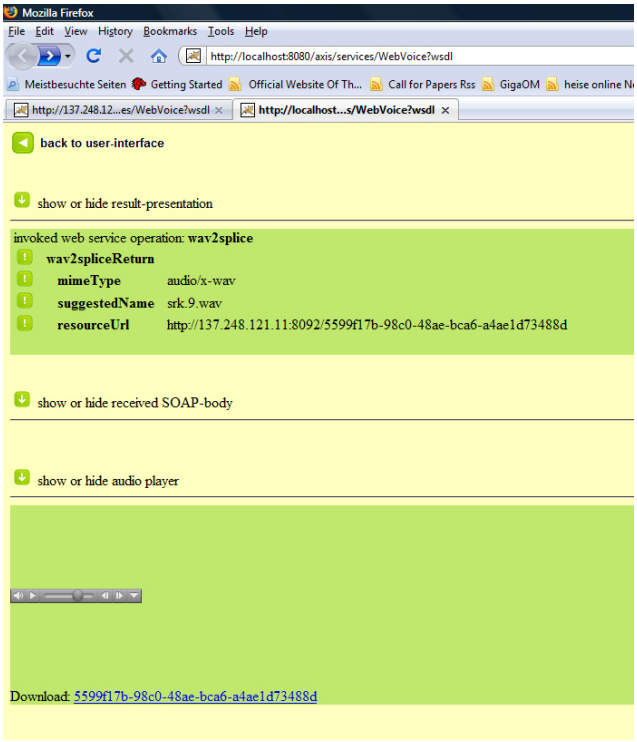

Figure 6. Result of the `wav2splice` operation.

In the first section of the result page, the data of the resulting SOAP message is shown. When the mouse slides over the white exclamation mark, the data type is shown as a tooltip. The second section is reserved for the original SOAP message (hidden in the screenshot). By clicking on the green "down" arrow, sections can be hidden and shown again. In the third section, a media player is embedded into the result page that plays the resynthesized audio file.

6 Related Work

Gemstone [1] is an application based on the Mozilla Application Framework and XULRunner that allows users to browse to a set of web services and enables dynamic integration of the user interface elements. The service providers have to specify the service and user interface. Gemstone is used to select services from a proprietary repository. Hence, Gemstone lacks a real browser integration. It only provides the integration of web services for which visualization code is written in XUL and JavaScript. This limits the use of Gemstone to service providers offering visualization code to their services and using the proprietary repository format supported by Gemstone.

Web and Grid portals like GridSphere (`http://www.gridsphere.org`) provide access to a collection of services and a set of tools such as single-sign on, data management, and certificate management or collaboration capabilities (sharing, interlinking, and integrating multidisciplinary data sets) like the GEON portal [8] directly through the browser. However, in order to operate portals, maintenance and administration efforts are needed. Furthermore, for each new service to add to the portal, a portlet has to be written.

The Web Services Remote Portlet (WSRP) specification [6] addresses part of the problem. A user interface defined at a remote site can be included in the portal. Still, each provider has to define the user interface for each of its service descriptions on the web or in a repository.

soapUI (`http://www.soapui.org`) is a popular test suite for inspecting, invoking, and developing web services. Unlike the Web Service Browser, soapUI does not integrate into the browser. It is used for testing purposes to assist the web service developer and not a service user. Furthermore, it does not provide the integration of different multimedia data types.

Although the Web Service Browser has many advantages compared to the related approaches presented above, it has the disadvantage that it depends on the Firefox releases of the Mozilla Foundation. Thus, possibly the plugin has to modified for major Firefox releases (like upgrading from Firefox 3 to Firefox 4).

7 Conclusions

In this paper, a Web Service Browser that automatically creates a user interface when a user browses to a WSDL file has been presented. To efficiently handle file transfers, an implementation of the Flex-SwA architecture has been integrated. Implementation issues dealing with the WSDL parser, the model and the HTML structure of the generated HTML page have been described. Two use cases have shown how the Web Service Browser works.

There are several areas for future work. For example, a notification system will be added to the browser, such that asynchronous service invocation is supported. A statistical analysis of user experience is planned. The support of REST services is planned. Furthermore, more result presentation styles will be added. Finally, an evaluation of the functionality in further use cases will be performed.

Acknowledgements

This work is partially supported by the German Ministry of Education and Research (BMBF) (D-Grid Initiative) and the Deutsche Forschungsgemeinschaft (DFG, SFB/FK 615, Teilprojekt MT).

References

[1] K. Bhatia, B. Stearn, M. Taufer, R. Zamudio, and D. Catarino. Extending Grid Protocols onto the Desktop using the Mozilla Framework. In *2nd International Workshop on Grid Computing Environments (GCE 2006)*, 2006.

[2] S. Heinzl, M. Mathes, and B. Freisleben. A Web Service Communication Policy for Describing Non-Standard Application Requirements. In *Proc. of the IEEE/IPSJ Symposium on Applications and the Internet (Saint 2008)*, pages 40–47. IEEE Computer Society Press, 2008.

[3] S. Heinzl, M. Mathes, and B. Freisleben. The Grid Browser: Improving Usability in Service-Oriented Grids by Automatically Generating Clients and Handling Data Transfers. In *Proceedings of the Fourth IEEE International Conference on eScience*, pages 269–276. IEEE Press, 2008.

[4] 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 *Proc. of the IEEE Int'l Conf. on Web Services*, pages 3–10. IEEE Press, 2006.

[5] J. Huff. Building firefox extensions. *Linux Journal*, 2007(160):8, 2007.

[6] A. Kropp, C. Leue, R. Thompson, C. Braun, J. Broberg, M. Cassidy, M. Freedman, T. N. Jones, T. Schaeck, and G. Tayar. Web Services for Remote Portlets Specification. OASIS Standard, August 2003.

[7] M. Mathes, S. Heinzl, and B. Freisleben. WS-TemporalPolicy: A WS-Policy Extension for Describing Service Properties with Time Constraints. In *Proceedings of the First IEEE International Workshop On Real-Time Service-Oriented Architecture and Applications (RTSOAA 2008)*, pages 1180 – 1186. IEEE CS Press, 2008.

[8] U. Nambia, B. Ludaescher, K. Lin, and C. Baru. The GEON Portal: Accelerating Knowledge Discovery in the Geosciences. In *8th ACM Int'l Workshop on Web Information and Data Management*, pages 83 – 90. ACM, 2006.

[9] D. Seiler, S. Heinzl, E. Juhnke, R. Ewerth, M. Grauer, and B. Freisleben. Efficient Data Transmission in Service Workflows for Distributed Video Content Analysis. In *Proc. of the 6th Int'l Conf. on Advances in Mobile Computing & Multimedia*, pages 7–14. ACM Press, 2008.