# WSInterConnect: Dynamic Composition of Web Services Through Web Services

Josef Spillner, Iris Braun, and Alexander Schill

Dresden University of Technology, Dept. of Computer Science, Chair for Computer Networks, 01062 Dresden, Germany
js177634@inf.tu-dresden.de, {iris.braun, alexander.schill}@tu-dresden.de

**Abstract.** In this paper, a model is presented which allows the composition of web services by means of a special web service, named WSInterConnect. Such a service might be used in a portal environment to allow interactive services lookup and creation of permanent composed services in the design stage, and for efficiently resolving missing parameters during the runtime of a process. The portal integration relies on augmented service description files based on WSGUI concepts [1], and provides a usable infrastructure for BPEL4People [4] concepts.

## 1 Introduction

The design of processes relying on web services has recently gained some attention in the area of human involvement. Specifically, several papers acknowledge escalation hooks for decisions which cannot be drawn autonomously. Likewise, the interaction with humans happens during design time, requiring facilities to compose and reuse web services dynamically. Somewhere in between, services attached to a process might have to be replaced at runtime.

WSInterConnect is a web service by itself which handles such simple composition models. It produces output information which is suitable for the actual execution of the composite service, for example, in a BPEL (*Business Process Execution Language*) engine. The way the service works is that it lets users create a sequence of operations based on WSDL files (*Web Services Description Language*), taking into account structural differences of the input and output messages of the service's operations.

It is however known that WSDL alone is not sufficient for describing web services for direct usage by humans and processes alike. This led to augmented service description efforts, namely WSGUI (*Web Services Graphical User Interface*) [2] for human interaction, WSDL-S (*Web Service Semantics*) to enable autonomous discovery and matching of web service operations, and others. The former will be referenced throughout this paper.

An example, which is modelled closely to existing BPEL4People ideas, is presented to demonstrate service composition, human intervention and inference of GUI elements by means of WSGUI. Integration into a portal environment, including a user database and a task management portlet, fulfils the requirements of the lifecycle of such a process as outlined in the BPEL4People draft.

F. Eliassen and A. Montresor (Eds.): DAIS 2006, LNCS 4025, pp. 181–186, 2006.

## 2 Web Service Composition

Traditional composition of web services happens either by creating the process descriptions by hand, or by using a graphical design tool. Often, these tools are easy to use but have severe limitations. In many cases, the message types of operations chained to each remain unchecked, resulting in type mismatch faults only at runtime. Additionally, a lot of overhead is generated by including such checks into the tool software, and compatibility is lowered by only supporting a restricted set of output formats. These shortcomings can be overcome by using a web service to handle the creation of process description files in various formats. The proposed service, WSInterConnect, will be described in detail in the next sections, beginning with a look on message comparison details.

Each web service operation takes one or more input parameters, and returns one or more output parameters, all possibly being of complex types, such that additional value restrictions, union types and lists can be represented. It is valid to assume a single parameter for input and output each, if both are considered to be complex on their own, forming a product type. In WSDL terms, a message might contain several message parts, but might also contain just one part which in turn consists of a sequence of elements.

Since WSDL data types are specified using XML Schema, a prerequisite for composing services interactively is to check for the compatibility of the output message of one operation with the input message of the other operation, with three possible outcomes for each check:

All-or-nothing compatibility: The messages are either fully compatible, or not at all. This can be trivially checked when only simple types are used, and recursively for complex types by parallel message comparison. An important note is that variable names might differ, only the type structure is important.

Subset compatibility: A sub-set relation can be specified, that is, one message is a subset of the other message. This includes list ranges as well as omissions of parameters on one side only. If the input of the second operation is a subset of the output of the first one, some variables will have to be discarded, otherwise some values will have to be injected, for instance by the user.

Arbitrary compatibility: Whenever none of the two cases above applies, the operations are considered to be incompatible. In such a case, WSInterConnect could reject to link them together in any form. An advanced implementation would however allow to both drop variables and let the user fill in missing values, as long as at least a certain percentage of compatibility is present.

Having identified the cases, the relevance of partial compatibility can be evaluated more easily.

In a fully automated services composition, only full operation compatibility can be allowed, while in an interactive environment such as a portal, the user can be the input source who determines the missing values of the input parameters of the second web service, in addition to the output of the first one. Likewise, if parameters occur in the output but do not have any matching counterpart in the following input, the user should allow to have them be dropped.

## 3 WSInterConnect

A web service which implements the concepts introduced above is called WSInterConnect. Its first functionality can be described as follows: For each pair of services, an operation is selected to serve as output of the first service and input of the second one, respectively. WSInterConnect will then create a composed *virtual* service, which requests from the user only those input variables which cannot already be filled out by the first service.

For more complex scenarios, rather than reinventing the wheel, existing process engines can be used to refer to WSInterConnect in order to request missing values. This is accomplished by the second functionality, resolving issues of missing parameters at run-time.

The first set of operations includes `compare()` and `store()`. The invokation and result of one WSInterConnect operation, `compare()`, which takes a number of WSDL URLs as its input, looks like about the following message conversation if the same WSDL is used for both input and output, effectively leading to a cartesian product of all operations offered by said WSDL:

```
<ConnectRequest>
 <wsout>http://localhost/mail.wsdl</wsout>
 <wsin>http://localhost/mail.wsdl</wsin>
</ConnectRequest>
<ConnectResponse>
 <comparisons>
  <comparison>
   <operation-output>signMail</operation-output>
   <operation-input>sendMail</operation-input>
   <status>compatible</status>
  </comparison>
  ...
 </comparisons>
</ConnectResponse>
```

The `store()` operation takes one of these matches and stores it as a composition for later use with a process engine.

Augmenting the WSDL for the WSInterConnect service, and the ConnectRequest operation in particular, with WSGUI information, makes it possible to generate a GUI which lets the user select the operations which are to be used for the composition. In addition, if *partial compatibility* is implemented, more detailed type mismatch information could be sent as to allow the structural alignment of input and output operation message format.

The basic WSGUI concepts shall be presented now briefly.

Pure WSDL files do not contain enough information to use web services as generic processes. Rather, specialised frontends have to exist to facilitate the navigation, data input, error handling and presentation of the results.

With WSGUI, it was shown to be possible to augment the WSDL files with user-centric visual hints, based on the XForms standard and some additional information like description texts and mime-type specifications.

WSGUI is an essential component in the realisation of interactively composed web services. To date there exist tools which attempt to handle this task without

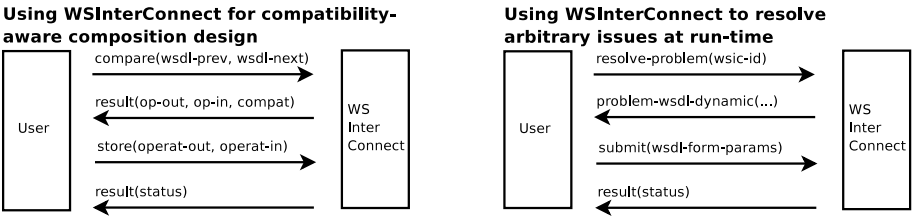


**Fig. 1.** Sequences describing usage patterns with WSInterConnect

WSGUI, but they are not up to current usability and internationalisation standards. Existing approaches like WSRP [5] are not able to adapt to fully arbitrary underlying data formats. Using the features of WSGUI, inferring GUI information from WSInterConnect is possible as follows.

There are two potential areas of WSGUI usage in combination with the service. First, the WSInterConnect service itself, by way of its `compare()` and `store()` operations, can be invoked dynamically by the user. This works like any other generic web service call in that its messages are rendered to a web page or client application. Second, the process as composed by WSInterConnect and executed by a process manager might call back so that extended WSGUI features such as marking certain fields (those which are already assigned a value within the process) as read-only, while permitting the user to fill out the remaining fields. Operations include `resolve-problem()` to retrieve the dynamically generated WSDL for a certain identifier, which has been added to WSInterConnect by the process, and `submit()` to submit the form which is based on the WSDL data, and rendered with WSGUI as described above. All major operations are shown together in figure 1.

## 4 Application Scenario

Following the often cited example of a travel agency, a public web service might exist which permits booking a flight, taking the four parameters name, destination, and suggested dates of flight and of return. A second service for booking the hotel rooms can reuse all of the data, but since the final destination city might not have an airport, the destination parameter is dropped and two new ones, for final destination and hotel category, are added instead, and it reuses the exact dates as returned by the first service.

This is a scenario where *partial compatibility* is involved. If for whatever reason the hotel booking service is not available, the user might on the fly substitute it with a compatible service. The substitute might however have a use for the flight destination city, for instance to provide a hotel room offer combined with airport shuttle service.

While this application scenario already suggests a possible use of WSInterconnect, the full power can be experienced when considering the following BPEL4People deployment scenario.

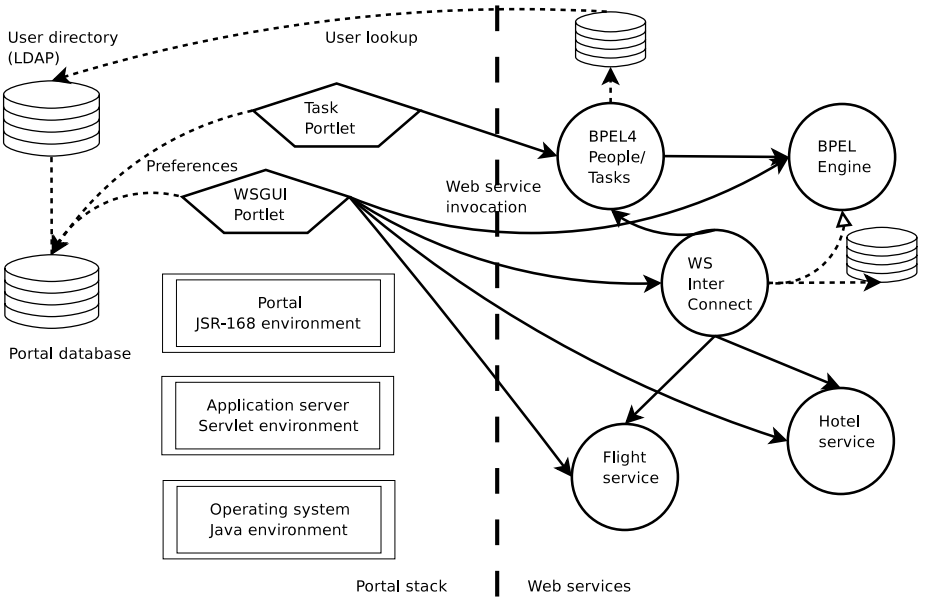


**Fig. 2.** WSInterConnect in a typical deployment of web services and a portal stack

Using web services in portals is a well-known concept for implementation of flexible service-oriented applications, for instance in telework scenarios [3]. The portal environment provides two necessary parts of the infrastructure: A user database, and a portlet container hosting various portlets. According to BPEL4People, web service escalation points are best suited for inclusion into a task tracker, which in this case runs as a service and is displayed in a dedicated portlet.

The architecture is outlined in the figure 2, distinguishing between the on-site portal stack and the web services.

The custom tracker portlet displays tasks with their priorities and deadlines, and contains hyperlinks to the WSGUI engine, parametrised with calls to the web services referenced by the BPEL4People engine so that clicking on the task presents an option list for resolving the task, which is automatically inferenced from the available options. This feature is using the `resolve-problem()` and `submit()` operations already introduced earlier.

When resolving a problem requires more results from other web services, the user could create new compositions on the fly from within the portal.

## 5 Related Work

There are a few tools which already handle the creation of processes based on web service composition, like X Workflow Composer [6] for direct process code generation, or commercial BPEL editors. They do however lack design-time

parameter comparison and methods to retrieve missing parameters from the user of the service at run-time. Graphical editors like Triana PSE [7] often focus on ease of use, an aspect not in the centre of WSInterConnect development, so that leveraging its advantages into the graphical editors at a later date seems to be an important proof of the concepts presented in this paper.

Until now, no effort which combines concepts similar to WSGUI and service composition is known to the authors.

## 6 Conclusion

It has been shown how to let users interact with web services in an interactive and dynamic environment, without the need to pre-define the operations or the visual appearance thereof.

WSInterConnect has been finished in parts. It is not expected that this service will lead to a standalone deployment. Rather, advanced web services concepts can be combined with WSInterConnect in a useful and still explorative manner.

Current WSGUI research hints at generating the needed GUIs automatically for even more usage scenarios, thus projects like WSInterConnect will be less of a novelty and more common technology in the near future.

## References

1. WSGUI concept project, J. Spillner, January 2006, `http://wsgui.berlios.de/`
2. *Creating GUIs for Web Services*, M. Kassoff, D. Kato and W. Mohsin, IEEE Internet Computing, September/October 2003, Vol 7, No. 4, 66-73, `http://logic.stanford.edu/~{}mkassoff/papers/wsgui.pdf`
3. *A service-oriented Architecture for Teleworking Applications*, I. Braun, A. Schill, Proceedings of IASTED International Conference on Internet and Multimedia Systems and Applications, Honolulu, August 2005, Acta Press, Calgary, p. 105-110
4. BPEL4People white paper, M. Kloppmann et. al., August 2005, ftp://www6.software.ibm.com/software/developer/library/ws-bpel4people.pdf
5. OASIS Standard: Web Services for Remote Portlets, Specification Version 1.0, (OASIS, 2003) `http://www.oasis-open.org/committees/wsrp`
6. X Workflow Composer, software project, S. Shirasuna, January 2006, `http://www.extreme.indiana.edu/xgws/xwf/`
7. *Triana as a Graphical Web Services Composition Toolkit*, S. Majithia et. al., Proceedings of UK e-Science All Hands Meeting, September 2003, EPSRC, p. 494-500 `http://www.trianacode.org/`