Integration of Heterogeneous Web Service Components

Xinjian Xu, Peter Bertok

School of Computer Science and IT, RMIT University, GPOBox 2476V, Melbourne, Victoria 3001, Australia

Abstract. Integrated value-added Web services can be developed by combining existing Web service components that are often heterogeneous in many aspects, such as having different interfaces and data encoding schemes. Web service integration also has to face the challenge that Web service components evolve frequently in response to business needs. This paper describes a three-layer-structured model for building reliable Web service compositions. The proposed layered structure can significantly reduce the complexity of integrating heterogeneous service partners. Furthermore, the evolution of a Web service component can be supported by making a rather minor amendment to the mapping file corresponding to this particular component.

1 Introduction

To provide value-added Web services multiple components may need to be integrated. However, different Web service components can be heterogeneous in many aspects: such as programming languages used, platforms, communication methods (synchronous and asynchronous), interfaces, data formats, data encoding schemes, content capabilities, supporting platforms, etc. In this paper we focus on two facets of heterogeneity: on capabilities, when different components provide different services, and on semantics, when some Web service components provide a similar, or exactly the same service.

Integration of service partners enhances a web-based service-oriented framework by providing a composite service over several existing services. However, building such a composite service can be difficult. Due to the dynamism and modularity of the environment in which Web service components operate, current models and solutions used in traditional component based distributed systems (CBDS) can not be applied to the Web services directly.

This paper presents a multi-layered model for integration of web services with heterogeneous interfaces. The model adopts interface conversion for building connection between Web service components. Method mapping, parameter mapping and return value mapping ensure good adaptability in dynamic environments. Our model divides the integration of heterogeneous Web service components into two sub-tasks: (i) building a layer of homogenized Web service components over the pool of components and (ii) building a composition on top of the layer of homogenized Web

Xu X. and Bertok P. (2004).
Integration of Heterogeneous Web Service Components.
In *Proceedings of the 2nd International Workshop on Web Services: Modeling, Architecture and Infrastructure*, pages 87-94
DOI: 10.5220/0002678600870094
Copyright © SciTePress

service components. The proposed method simplifies web service integration considerably, both in static and in dynamic environments.

The next section gives an overview of existing approaches to web service integration, and briefly explains different approaches. It is followed by a description of the proposed model, and implementation details are also provided. Finally, discussion and conclusion are presented.

2 Existing Work

There have been two different approaches to web service integration. The first approach adjusts the relationships between components without breaking them. The most frequently used solutions with this approach are the following [1].

1. Manipulating the parameters of components to provide variant services.
2. Providing multiple classes of service.
3. Re-customizing the services according to user profiles
4. A dedicated object takes care of the integration of components.

With this approach adaptation to changes is faster, as there is no need to establish new trust relationships. Redundancy is also reduced when components' basic functionalities are the same but constraints are different. The price for this better run-time performance is paid at service creation time, as more comprehensive specification of components is needed. The approach offers only limited fault tolerance, as unavailability of a particular service cannot be predicted at service creation time.

The second approach breaks up the relationship between the composed components and performs re-composition by rebinding [1]. This can be performed in the following ways.

5. Replacing one component at a time, such as replacing the old component with a new component. In this sense, the composition structure is relatively static.
6. Breaking the composition structure by replacing two or more components at a time.

This solution solves the problems from a different point of view than the previous approach. It stores multiple integration plans that offer the flexibility of selecting the most suitable Web service components, because one sub-request could be served by one or more Web service components. This approach also supports dynamism, as the selection decision will not be made until the last minute. However, this solution also has some disadvantages. Similarly to the previous solution, fault tolerance is also a problem here as outsource agents do not check the availability of the selected Web service components and the invocation of a selected Web service component can fail due to the unavailability of that component.

3 Proposed Model

3.1 Issues

Providing interoperability between Web service components is a more difficult task than it is in a traditional Component Based Distributed environment because the interaction between Web service components is driven by a business logic that is affected by market or business objectives, and thus can change from time to time. This leads to possibly dynamic connections between Web service components and can affect their behavior. To cater for these, several issues need to be addressed when integrating service partners.

- Compatibility with current XML-based standards
- Support for Web service component dynamism
- Dealing with semantic differences
- Handling service capability differences
- Quality of Service (QoS)

3.2 Structure of the Proposed Model

The proposed model employs a layered structure to reduce the complexity of the solution. It separates the integration of Web Service components into two sub-tasks, as integrating semantically equivalent service components and integrating semantically non-equivalent service components are performed in different layers. The model has three layers: pool of Web service components, service collector layer, and composite service collector layer. The structure of the model is shown in Figure 1.

In this model, application programs interact with the layer of composite service collector, which provides a composite service on top of the service collector layer. The homogenized interface for the top layer is produced by a set of service collectors via performing an interface conversion for the several semantically equivalent Web service components below. To perform the task, this layer contains relevant mapping information about interface conversion and also contains relevant mechanisms to handle the key issues discussed in the previous section.

SciTePress Science and Technology Publications

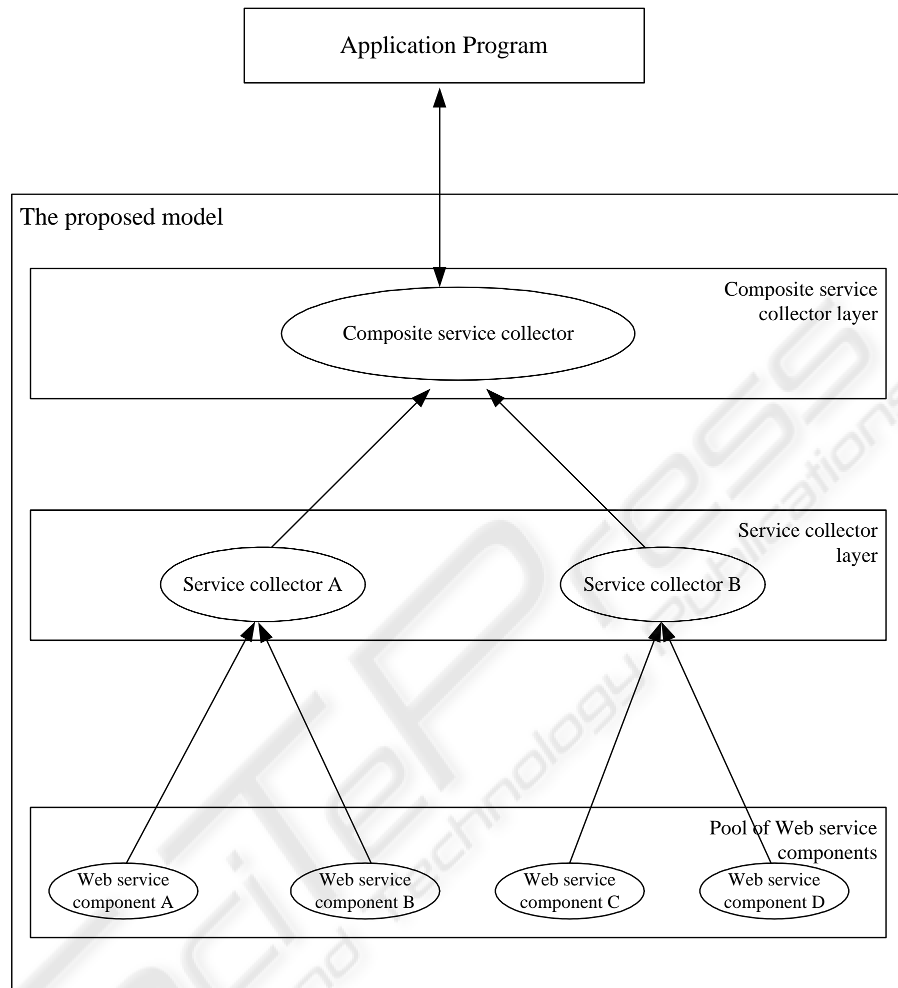

Fig. 1. Structure of the proposed model

3.3 Details of the Proposed Model

The Pool of Web service components
The Web service components form the bottom layer of the model. They services are distributed across the Internet. Each component is an individual application that does not rely on any other service components. We assumed that every Web service component had a WSDL defined interface, regardless of the programming language used to implement it. The services are registered with UDDI, which accepts SOAP query messages and returns the description (including the location) of the required

component. The Web service components can be heterogeneous in many aspects. The proposed model, however, addresses only heterogeneity in their interfaces.

The Service Collector Layer

Interface heterogeneity is handled mainly in this layer. The service collector layer consists of multiple service collectors, which aggregate semantically equivalent or alternative components. The service collectors themselves are semantically non-equivalent, since there are no two service collectors providing similar or the same services in the proposed model.

The major responsibility of a service collector is to implement interface conversion by building a homogenized interface based from the interfaces of the aggregated components. It is performed by mapping the interfaces of aggregated Web service components to a new interface, as shown in figure 2. A service collector has the following components: a service collector coordinator, mapping file(s) and a description file.

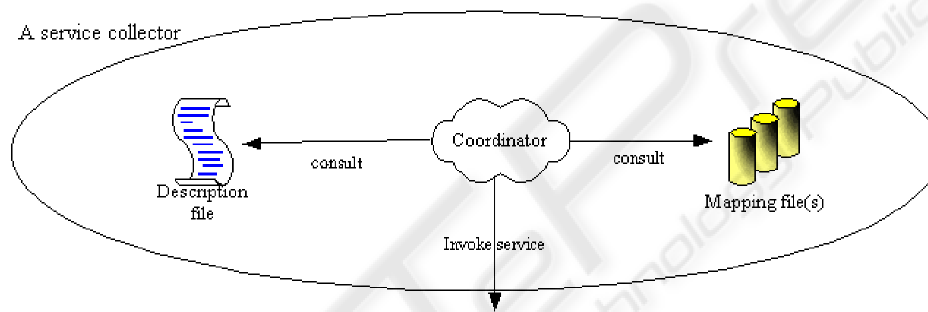

Fig. 2. Components of a service collector

The service collector coordinator

The service collector coordinator communicates with the upper layer, accepts requests from there and sends the results back. It also selects the most suitable component to serve the request, based on the content of the request and the service description file of the aggregated components. Based on the relevant mapping file, it converts the interface of the service collector to the interface of the selected Web service component. The service collector also communicates with the lower layer, invokes the selected components and accepts the returned data. It also interacts with the coordinator of other service collectors, if the invocation of other service collectors depends on the result of invoking the current service collector. Finally, it exchanges messages with the UDDI service to retrieve the WSDL interface of the selected Web service component.

Mapping file(s)

There is a map file created by the "implementer" for each aggregated component respectively to do the mapping between the interfaces dynamically. When the service

collector coordinator selects a specific Web service component, it loads the relevant mapping file.

Description file

This file, also in XML format, contains specification about the aggregated components. The specifications consist of the information about the service component, including service description, name, location, interface data, and rank.

WSDL interface vs. Java Interface

In addition, there are two options to create the interface of a service collector. One is to create a WSDL interface and a Java interface, the other is to only build a Java interface. The advantage of the availability of a WSDL interface is that the service collector can also advertise its service through the UDDI server. Therefore, external users can access the service collector directly (no need to use the service of the upper layer). In contrast, the advantage of having only a Java interface is that the mapping between a Java interface (of the service collector) and a WSDL interface (of aggregated Web service components) is simpler.

The Composite Service Layer

The composite service layer is built on top of the Service collector layer. It provides the end user with general access to its composite service, which aggregates multiple service collectors. Compared with the aggregation in the service collector layer, the aggregation in this layer is more straightforward. Since different service collectors provide different services, the likelihood that their services may overlap is low. Therefore, the composite service layer just extracts interface data from all the service collectors and "glues" them together.

In this layer, metadata about all of the aggregated service collectors is used. Metadata describes capability, location and access information of services collectors. Therefore, this layer contains the following components: a composite service coordinator, metadata storage, and an execution model file.

The composite service coordinator

In the top layer the coordinator decomposes the requests into several sub-tasks. Then it assigns each sub-task to a service collector for processing, after checking the stored metadata. In the next step it invokes service collectors via exchanging information with the service collector coordinator. Finally, it synthesizes the results from collectors and returning the synthesized results to the calling application.

Metadata Storage

This storage file serves the same purpose as the description file residing in the service collector layer. It describes capability, location and access information of the services collectors. The coordinator uses these metadata to locate, browse and invoke the service collectors.

Execution model file
This file is about how to compose service collectors and is created manually. According to this file, the coordinator invokes the service collectors in the format of sequence, parallel, or combined. It can be also presented in an XML file format.

Implementation
To test the feasibility of the proposed model, the proposed model has been implemented in Java programming language under JWSDP1.2. Our implementation has concentrated on the interface conversion, which is carried out by the service collector layer.

4 Discussion

The proposed model is compatible with current XML-based technology. In our solution, each Web service component has a WSDL interface, which is accessible from the web and published on the UDDI registry server. In this way, the requester of a Web service can first query with the UDDI registry server for the location and description of the requested Web service component by using JAXR APIs, and then invoke the Web service component by using JAX_RPC APIs. The middle layer does not know which Web service components will be selected until runtime since it has to dynamically make the selection based on the selection policy and the data input by the end user. Consequently, a dynamic invocation interface (DII) had to be adopted by the service collector that acts as a DII client. The invocation between the top layer and the middle layer is different, since the location and interface of the service collectors are relatively static. Each service collector has a stub object, which is generated by "wscompile" from the WSDL interface of the service collector and resides in the top layer. This local stub object acts as a proxy for the service collector as shown in figure 3. In our implementation, a dynamic proxy client is created for the top layer to interact with the middle layer.

In the proposed model, the change made to the interface and location of one Web service component does not have any significant affect on the integration of service partners; all we have to do is to make a rather minor amendment to the mapping file corresponding to that particular Web service component.

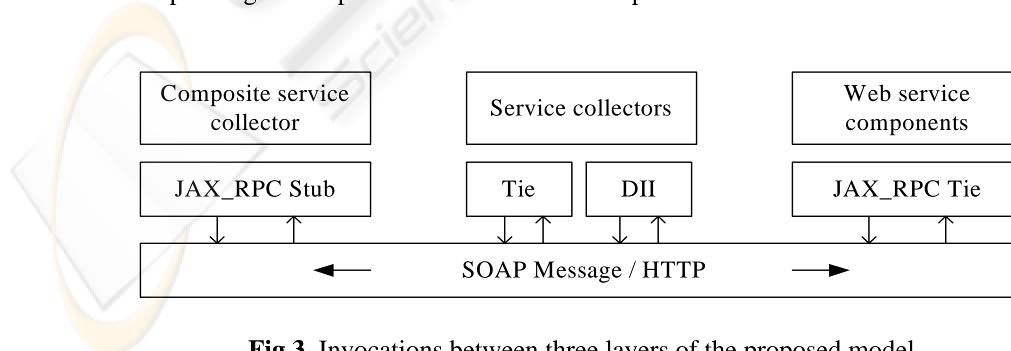

Fig.3. Invocations between three layers of the proposed model

5. Conclusion and Future Work

Our implementation has shown that the proposed solution is feasible and can be implemented in JWSDP. Interface conversion proved to be a stable and efficient method to build connections between components in an ad-hoc environment. With the layered structure, the task of providing a composite service is decomposed into two sub-tasks: (i) creating a service collector layer, to homogenize the semantically equivalent Web service components, and (ii) aggregating the different service collectors, and providing a common access to service requesters. During runtime, the coordinators of the service collectors can dynamically select a Web service component and perform the conversion by using mapping files.

Security is not addressed by the model described here, it will be addressed in future extensions.

References:

1. V. Tosic, B. Pagurek, B. Esfandiari and K. Patel, "*On Various Approaches to Dynamic Adaptation of Distributed Component Compositions*", Technical Report OCIECE-02-02, Ottawa-Carleton Institute for Electrical and Computer Engineering (OCIECE), June 2002
2. Benchaphon Limthanmaphon and Yanchun Zhang, "*Web Service Composition With Case-Based Reasoning*", in Proceedings of the 14th Australian Database Conference (ADC2003), Conferences in Research and Practice in Information Technology, Vol. 17, Page(s): 201-208, 2003
3. Francisco Curbera, Ignacio Silva-Lepe and Sanjiva Weerawarana, "On the Integration of Heterogeneous web service partners", in Proceedings of the OPPSLA 2001 Workshop on Objected-Oriented Web Services (OOWS2001), 2001, http://www.research.ibm.com/people/b/bth/OOWS2001/curbera.pdf
4. Paolo Predonzani, Alberto Sillitti and Tullio Vernazza, "*Components and Data-Flow Applied to the Integration of Web Services",* Industrial Electronics Society, 2001. IECON '01. The 27th Annual Conference of the IEEE, Volume: 3, 29 Nov.-2 Page(s): 2204 – 2207, Denver, USA, Dec. 2001.
5. Paulo F. Pores, Mario R.F. Benevides and Marta Mattoso, "*Mediating Heterogeneous Web Services*", in Proceeding of the 2003 Symposium on Applications and Internet Workshops (SAINT'03 Workshops), pp 344 - 347, Orlando, USA, Jan. 2003.
6. Peer Hasselmeyer, "*Managing Dynamic Service Dependencies*", in Proceeding of 12th International Workshop on Distributed Systems: Operation and Management DSOM'2001, pp 141-150, Nancy, France, 2001