

Loose Interface Definition: An Extended Interface Definition for Dynamic Service Composition

Keita Fujii (kfujii@ics.uci.edu) and Tatsuya Suda (suda@ics.uci.edu)
Dept. of Information and Computer Science,
University of California, Irvine
Irvine, CA 92697-3425

Introduction: Dynamic Service Composition

The concept of Dynamic Service Composition is to create a complex composite service through combining multiple software or hardware components at runtime. Dynamic Service Composition is similar to object-oriented or component-based software design approach since in both approaches an application is built as a combination of multiple components. However, in Dynamic Service Composition, an application is composed dynamically and autonomously by discovering, selecting, combining and organizing multiple components at runtime, whereas in the traditional object-oriented approach the component selection and organization are carried out by application designers at the implementation phase. This dynamic composition feature of Dynamic Service Composition provides flexibility and adaptability to applications. For example, an application built on top of the Dynamic Service Composition system is able to change its user interface dynamically according to user's preference (e.g. English/Japanese menu, colorful/simple buttons etc.) or hardware configuration (e.g. low resolution for PDA, high resolution for PC) by permutating its user interface components. In the Dynamic Service Composition system, it is also possible to modify functionality or performance of an application to adapt various resource constraints or configurations (CPU speed, network bandwidth etc) by selecting optimum components and organizing them properly. Furthermore, a totally new application may emerge by combining several components designed for entirely different purposes. The Dynamic Service Composition system must be intelligent enough to be able to find an optimal solution of composing a logical, useful and functional application that satisfies given requirements under various conditions and limitations. It must also be autonomous so that an application could be composed with minimal user inputs. Several research projects have recently been done to realize such architecture.

In this paper, we propose Loose Interface Definition, a set of techniques for defining interfaces of components. Loose Interface Definition maximizes the possibility that two different and independent components can communicate with each other. In the rest of this paper, we first explain how interface definition is used in Dynamic Service Composition. Next, we illustrate the problems of using existing Interface Definition Languages in Dynamic Service Composition. Then, we introduce Loose Interface Definition, explain its importance in Dynamic Service Composition, and prove its effectiveness onto the existing Web Service. Lastly, we propose XML Schema to describe Loose Interface Definition.

Dynamic Service Composition by Interface Matching

One of the possible mechanisms to compose an application from multiple components is to combine them sequentially such that the execution result of the former component is passed to the next component and accepted as its input. The emerging sequential composite service can provide an output from a given input that none of the existing components can provide by itself. An example of such sequential composite services is a chat application, composed as a sequence of an input device component (keyboard or microphone), a network component and an output device component (display or speaker).

Notice that neither input, network or output component could provide the chat functionality as an independent component.

In order to organize a sequential composite application dynamically, the system should be able to discover or detect pairs of components such that the output of one component is equal or equivalent to the input of another. Under the assumption that each component has a set input and output interface definitions describing the message formats it can accept or generate, discovering such pairs can be replaced by matching an input and an output interface definition of two components. Once such pairs are identified, a sequential composite service can be easily constructed by appending a pair after another which contains the same component. Be aware that interface matching can be applied not only to sequential service composition but also to more complex composition scheme.

Problem of existing Interface Definition Language

Although there exist several Interface Definition Languages such as CORBA IDL¹, MIDL² and WSDL³, they are not adequate for Dynamic Service Composition, because they assume that two components must have exactly the same interface derived from the same interface definition in order for them to be able to communicate. This assumption may either restrict the variety of services, or restrain the possibility of communication. Suppose that a company wants to implement a flight reservation service using its advanced reservation system that allows a user to choose a seat location and meal plans. However, if the interface definition used by other flight reservation systems only includes traveler's name, travel date, departure and destination location, the company cannot take advantage of its advanced reservation system, because no information regarding seat and meal preference is available in the interface. This example shows that sharing the same interface definition may restrict the variety of services. The alternative choice for the company is to design its original interface definition so that it can take full advantage of its reservation system. However, this choice may restrain the possibility that other components can access to the service. At the time the company deploys its service, nobody knows its original interface definition. Also it is very unlikely that other components happen to have an exact same interface. Consequently, no (or little) components can communicate with the company's service.

Our Solution: Loose Interface Definition

In order to maximize the possibility of communication without losing the variety of services, we propose Loose Interface Definition. Loose Interface Definition is a set of techniques for defining interfaces of components. Even though two components have different interfaces, if the interfaces are defined as 'loose' (as explained later), and are defined for the same content (e.g. both implement 'flight reservation service'), there is relatively higher chance that those two components can exchange messages and communicate.

An interface of a component is usually defined as a set of elements, each of which consists of its name and type (integer, string etc). This interface definition style is similar to the method definition in object-oriented language or function definition in procedural language, where a method (function) is defined as a set of variable definitions.

Loose Interface Definition is a set of three techniques: allowing multiple names for an element, mapping elements by their names, and allowing optional elements. Allowing multiple names for an

¹ Common Object Request Broker Architecture Interface Definition Language: <http://www.corba.org/>

² Microsoft's Interface Definition Language:

http://msdn.microsoft.com/library/default.asp?url=/library/en-us/midl/midlstart_4ox1.asp

³ Web Service Description Language: <http://www.w3.org/TR/wsdl>

element solves the ontology problem, where two synonyms are recognized to represent different contents. For instance, human beings can easily understand that 'surname' and 'last name' mean the same content, but machines using simple keyword matching cannot. Loose Interface Definition solves this problem by allowing multiple names to be associated to a single element. Therefore, an element representing a last name is allowed to have both 'surname' and 'last name' as its element name, so that it can be recognized with either keywords. Mapping elements by their names simply means that the order of the elements is ignored when comparing two interfaces because it is irrelevant to their contents. Ignoring the order of elements resolves the inconsistency of elements' order. For example, a name of a person can be described either as 'last name, first name' or as 'first name, last name,' but no content difference exists between two representations. Allowing optional elements enables components to define original interface definitions in order to take full advantage of their services without losing compatibility to other interfaces. For instance, in the previous example of flight reservation service, the company can implement an original interface for its advanced reservation service that is still compatible with others by defining seat and meal preferences as optional.

By using Loose Interface Definition, two interfaces are recognized as compatible even if they are defined with different synonyms, in different order, with optional elements. These techniques improve the possibility of communication between two components that are implemented individually by two different designers.

Verification

In order to verify the effectiveness of Loose Interface Definition, we conducted a simple measurement using existing Web Service. Web Service is a general term of a set of technologies (UDDI⁴, WSDL, SOAP⁵) to discover, integrate and communicate service components via XML/HTTP. We first gathered the interface definitions of available Web Service components from UDDI repository, and calculated the number of pairs of interfaces that are exactly the same. Then we applied the techniques of Loose Interface Definition onto those interfaces, calculated the number of pairs of compatible interfaces again, and compared the result with the one before applying the techniques. As a result, applying Loose Interface Definition doubled the number of pairs of compatible interfaces. This result indicates that Loose Interface Definition is effective on increasing the possibility of communication of the current Web Service components.

Implementation

We propose that XML Schema could be one of the applicable methods to implement Loose Interface Definition. XML Schema is a schema definition language for XML, proposed by W3C. XML Schema is suitable because it is possible to describe all the techniques of Loose Interface Definition in XML Schema. Those three techniques (allowing multiple names for an element, mapping elements by their names, and allowing optional elements) are described with *<substituteGroup>*, *<all>*, and *<minOccurs>* elements in XML Schema, respectively.

XML Schema also has several useful features for defining an interface, such as describing acceptable patterns of elements (e.g. regular expression for string element, or range of value in numerical element), or inheriting and extending existing schemas. There are already many tools and libraries available for XML Schema that we can take advantage of. We have been developing a simulation for dynamic service composition using XML Schema and Java.

⁴ Universal Description, Discovery and Integration: <http://www.uddi.org>

⁵ Simple Object Access Protocol: <http://www.w3.org/2002/ws/>