Proposal: Protocol Oriented Service Deployment Platform

Anis Benyelloul
anis.benyelloul at gmail.com

June 2008

Abstract

Web services represent a first step towards seamless and easy application integration. *Web service protocols* represent an enhancement over web service interface description, that describes all the possible sequences of expected and sent messages of a service. Most existing tools for leveraging service protocols are design-time tools. The present work introduces a service deployment platform that provides *runtime* support for service protocols. The platform is being developed on top of Apache Axis2 web service implementation.

1 Introduction

The main purpose of SOA is to interconnect independent applications in order to build new ones. One of the most important issues that arise is how to make sure that two different applications, that were developed independently and that run on two different platform are going to interoperate.

Web services are a first step towards the solution. Web services' purpose is to standardize the lower levels of communication to make sure that messages exchanged between two services always follow the same format and data representation (SOAP) and to defines the messages that a service can receive/send as well as the corresponding arguments number and types (WSDL).

What web services do not address is how a service behaves *dynamically*. For example, if a web service accepts an operation "`cancel`" but only at certain points during the interaction, there will be no way (except from a documentation in a natural language) to guess it from any web service artifact.

This is what *Service Protocols* try to do: Describe how an interaction with a service behaves *dynamically*. In other words, at each point in an interaction, the set of all possible messages a service *might* send, and the set of all possible messages a service *expects* to receive.

The most common model for service protocols is *Finite State Machines*(FSM). The states are the abstract individual states a service can be in, while transitions represent messages sent or messages received (the sending or reception of a message is what makes a service move from one state to another).

All in all, a service protocol can be seen as an *extension* to the WSDL specification, that gives higher level information about how to *use* a service.

2 Existing Work

Many past projects (c.f [1, 3, 2]) tried to exploit the notion of service protocols to implement tools that support service implementation and protocol compatibility check. Following is a non-comprehensive list of work on the subject:

Protocol operations Given two protocols (FSM):

- Say if they are compatible;
- Say if they are partially compatible (if there are some execution scenarios where they may interoperate);
- Say if they are replaceable (if you can replace one service protocol with another without breaking clients of the service);

- Compute the intersection of two protocols;
- Compute the difference between two protocols.

Implementation verification :

- Given a service implementation, say if it conforms to a specified protocol. For example we could imagine that all book sellers would agree on one standard protocol for selling books, and than new book sellers could check their implementation against the standard protocol;
- Given a service protocol to conform to, generate a code template for implementing the service;
- Given a trace log of an interaction, extract as much information as possible about the protocols of the two services

3 Proposal

Most of the existing support for protocol oriented service design takes the form of *design time* tools.

The proposal is to create a *Protocol Oriented Service Deployment Platform* whose objective is to provide *runtime* support for protocol oriented service design.

The following two sections explain the service model around which the platform revolves and the specific *runtime* support it can offer to developers.

3.1 Service Model

In this model, we consider that each service is a black box that needs to communicate with other services. For each remote service it needs to interact with a service exposes an "*interface*". An interface defines:

- The set of all messages (method names and arguments) that a service might send and receive (e.g using WSDL);
- The protocol that this interface conforms to. In other words, the specification of the messages that can be sent to the interface, and the messages that the service can send through it, at each point during an interaction.

Protocols are modeled as finite state machines. Each protocol starts in an initial "begin" state. Transitions represent either a message being sent or received. The set of all transitions going out of a state represent all the messages that the service might send/receive in that state. When a message is sent or received the corresponding transition is activated and the protocol moves to the next state, if such a transition does not exist a "protocol conformance error" is generated.

We do not make the distinction between "server" services and "client" services. We consider that both have to expose an *interface*, and that the interaction will occur when the two interfaces are "connected". No one has to be thought of as the "client service" or the "server service", once the interfaces are connected communication is symmetrical and bidirectional.

On important aspect is that the actual connections (bindings) between interfaces are not encoded in the service implementation itself, but are provided as a separate deployment description file.

3.2 Runtime support

The actual contribution of this work is a service deployment platform that provides *runtime* support for the previously described service model. Specifically the support consist of:

API The platform comes with an API that facilitates the development of services according to the model;

Protocol Enforcement The platform keeps track and maintains the current FSM-state for each interface of the service. This permits to filter non-expected incoming messages at the *middle ware* level, without having to write unexpected message handlers in the service code itself. The platform also (optionally) filters out going messages to make sure they are in conformance with the exposed protocol.

Synchronization The platform makes sure both sides of an interaction *see the same sequence of*

message exchange, in order for the protocols to remain synchronized. This is important because the networks are asynchronous by nature, and we can have both services send a message at the same time, in which case they may see a different message sequence (e.g they may both think their their own message was sent before the other one's). So they may no longer agree on the current interaction point.

Interface Exposure The platform exposes the interface of the deployed services for potential service users. Additionally, upon deployment, the platforms checks if one of the remote services support protocol interfaces (e.g it is deployed on another instance of the platform), and if it's the case downloads the remote protocol and checks if it's compatible with the local one.

4 Implementation

The platform is currently being developed on top of Apache Axis2 web service implementation.

References

[1] Benatallah, B., Casati, F., and Toumani, F. Web service conversation modeling: A cornerstone for e-business automation. *IEEE Internet Computing 8*, 1 (2004), 46–54.

[2] Benatallah, B., Casati, F., and Toumani, F. Representing, analysing and managing web service protocols. *Data Knowl. Eng. 58*, 3 (2006), 327–357.

[3] Yellin, D. M., and Strom, R. E. Protocol specifications and component adaptors. *ACM Trans. Program. Lang. Syst. 19*, 2 (1997), 292–333.