Facilitating Complex Web Service Interactions Through a Tuplespace Binding

Daniel Wutke, Daniel Martin, and Frank Leymann

Institute of Architecture of Application Systems
University of Stuttgart
Universitaetsstrasse 38, 70569 Stuttgart, Germany
{wutke,martin,leymann}@iaas.uni-stuttgart.de
http://www.iaas.uni-stuttgart.de

Abstract The SOAP messaging framework, as one key technology of the Web service technology standard stack, defines a standardized message format for Web service interactions, a set of rules governing their processing and a mechanism that describes how SOAP messages can be transmitted over different network transport protocols, called *SOAP bindings*. The most prominent example for a Web service transport today, is the Hypertext Transfer Protocol (HTTP), which however suffers from certain drawbacks such as being inherently synchronous in nature and not providing decoupling of message sender and receiver in reference or time. In this paper, we present tuplespace technology as an alternative Web service transport that is characterized by a number of properties that are not found in current Web service transports: asynchronism, strong decoupling of sender and receiver and support for advanced message exchange patterns, such as one-to-many interactions, directly on the transport level. We describe the representation of SOAP messages in tuple form and exemplify how to use the operations provided by the tuplespace interface to realize certain Web service message exchange patterns.[1]

Key words: Web Services, Message Exchange Patterns, Tuplespaces, Web Service Binding

1 Introduction

Web service technology has gained broad acceptance in research and industry due to enabling loosely coupled interactions between communication partners which can be conducted over potentially multiple different network transport protocols while retaining end-to-end quality of services. Web services are defined by a set of specifications that enable standards-based service description, discovery, invocation, and composition through the use of WSDL, UDDI, SOAP, BPEL and others.

[1] This work is funded by the European Commission under the TripCom project (IST-4-027324-STP).

The SOAP messaging framework [1] defines a standardized XML-based message format and a set of rules that govern how SOAP processing nodes along the message path from initial sender to ultimate receiver should process a SOAP message. In addition, SOAP defines a mechanism to bind SOAP messages to different network protocols to enable their transmission between SOAP processing nodes over a network through so-called *SOAP bindings*. For this purpose, they define a serialization of the SOAP infoset in such a way that it can be transmitted by a sender over the chosen network transport protocol and reconstructed by the receiver (or the next hop/node in case of multi-hop interactions). Furthermore, they describe how the services of the underlying transport protocol (i.e. its interface) are used to transmit the chosen serialization of the SOAP infoset between SOAP processing nodes and describe potential failure scenarios that can be anticipated within the binding.

Tuplespaces have their origin in the *Linda coordination language*, defined in [2] as a parallel programming extension for programming languages for the purpose of separating coordination logic from program logic. A tuplespace is conceptually similar to a piece of memory shared among all participants of an interaction which provides clients with synchronized access to *tuples* (i.e. an ordered list of typed fields) via a simple interface: tuples can be stored (using the *out* operation), retrieved destructively (*in*) and retrieved non-destructively (*rd*). Tuples are retrieved associatively using a template matching mechanism, i.e. by providing values of a subset of the typed fields of the tuple to be read.

The remainder of the paper is organized as follows: first we motivate the work presented by elaborating on certain unique properties of tuplespaces when compared to existing Web service transports (Section 2). Subsequently, the SOAP binding for the tuplespace transport is presented, consisting of (i) a description of how the information contained in a SOAP envelope can be mapped to tuples to facilitate their transmission over a JavaSpaces transport (Section 3) and (ii) how message exchange patterns can be mapped to Linda communication primitives (Section 4). Section 5 concludes the paper.

2 Tuplespace binding for Web services

As of today, HTTP [3] is still the most widely accepted Web service transport. Due to the nature of HTTP being designed for direct, synchronous client-server interactions, it shows certain drawbacks with regard to decoupling sender and receiver in reference and time. As a result of tight referential coupling when conducting Web service interactions over HTTP, the sender of a message is required to explicitly address the concrete address (also referred to as *endpoint*) where a particular service implementation can be reached. If the location of the service implementation changes, a corresponding change has to be performed on the client. Furthermore, due to HTTP not offering decoupling in time, both message sender and receiver have to be available at the same time. If e.g. the receiver of a Web service invocation request is not available at the time of request sending, the message cannot be received and thus not processed. To overcome aforementioned shortcomings with regard to decoupling message sender and receiver in reference and time, a number of SOAP bindings have been proposed that build on messaging network transport protocols such as SMTP, XMPP or JMS which all employ the mechanism of store-and-forward to achieve decoupling in time (the sender hands over the message to transmit to the messaging system which delivers it as soon as the "next hop" becomes available). In addition, message recipients are addressed by logical identifiers instead of concrete addresses (e.g. an e-mail address in case of SMTP or a queue/topic name in JMS) which enables referential decoupling.

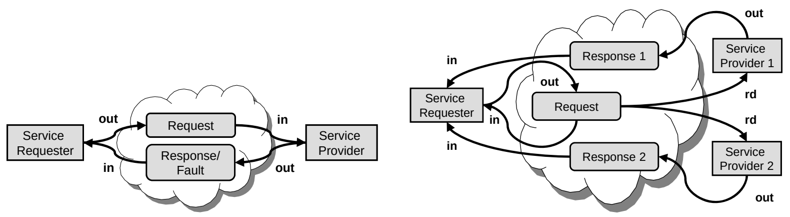
Although tuplespaces are in their use and behavior somewhat similar to messaging technology (see e.g [4] for a comparison), they are characterized by certain unique properties. In contrast to message-oriented middleware, where sender and receiver communicate by exchanging messages over queues and topics identified by logical addresses, in tuplespaces data is exchanged by senders publishing the data they want to communicate to a shared space (the counterpart to a queue/topic in messaging) on which potentially multiple receivers are listening. Data is consumed by the receiver in an associative manner, meaning that the receiver of a data tuple describes its content by example, e.g. the data types of certain tuple fields or their value. As a result, tuplespace-based communication is based on a *pull* mechanism (i.e. the receivers actively select what they want to receive by template-based consumption of tuples) as opposed to a *push* mechanism employed in messaging (i.e. the sender addresses a certain queue/topic, from which it expects the receiver to consume). Furthermore, in tuplespace-oriented communication, data is regarded as a published object instead of a message directed to a certain receiver. This means that when a sender publishes a piece of information to a tuplespace, the (one single) data tuple is available for all receivers listening on the tuplespace (which in particular also includes the sender of the tuple). This enables certain communication patterns that are difficult to realize based on other – e.g. messaging-based – Web service transports such as the Request-for-bid pattern described later in the paper. For instance, after publication of a message, the sender of the message can destructively consume the message again, update its contents and re-publish it. In addition, data can easily be directed to a set of receivers (similar to broadcast/multicast communication) or processed by a set of potential competing consumers in style of the *replicated-worker* pattern [5].

3 Mapping SOAP messages to tuples

To enable communication of SOAP envelopes over a network transport protocol, (i) the SOAP envelope to be transmitted needs to be encapsulated in an object that can be transmitted using the communication primitives of the respective transport protocol and (ii) information necessary for message identification, delivery and correlation has to be added and made accessible to evaluation by the transport. In case of tuplespace-based communication, data is encapsulated in tuples; the individual tuple fields are defined as follows. The field identified by the *Content* property contains a representation of the actual SOAP envelope, comprising both SOAP headers and SOAP body. The encoding of the SOAP message is specified by the *Content type* property. In most cases, the preferred content type is "application/soap+xml" where the SOAP message is transferred as a XML string in UTF-8 encoding; however other encodings are possible such as e.g. "multipart/related" in case of a MIME encoded SOAP message with binary attachments. The *MEP* property identifies the message exchange pattern that governs the exchange of the respective message. Possible values for this property are e.g. the identifiers of the WSDL 2.0 specification [6]. Each tuple with an encapsulated SOAP message is uniquely identified via the *Message ID* property. If the SOAP envelope contains a WS-Addressing [7] header block, the WS-Addressing Message ID is propagated to the tuple level to allow its use for template matching. To enable correlation of messages as part of interactions that involve more that one message exchange between communication partners, the *Correlation ID* property of a message can contain the Message ID of another message which is "in relation" to the given message. How a message relates to the message with the given correlation ID is defined through the *Relationship type* property. Valid values for the relationship type property are dependent on the message exchange pattern the message belongs to; WS-Addressing relationship type values are reused where possible. To enable addressing one particular Web service provider in case multiple Web service providers are connected to the same tuplespace, the *Service name* property must contain the name of the destination service in form of a Uniform Resource Identifier (URI). To allow service providers to correctly dispatch incoming SOAP messages to a service implementation (if the service provider for instance offers more than one operation), the *SOAP action* property conveys the semantics of the SOAP message in the content property in form of a URI. The *Binding version* property describes the version of the binding to be used to allow for further development and extension of the binding while still retaining backwards compatibility in implementations. In case the message encapsulated in the content property is a SOAP fault, the *Is fault* property contains a boolean *true* value, otherwise it has the value of a boolean *false*. Propagating this information to the tuple level enables e.g. convenient retrieval of all fault or non-fault messages. In case a SOAP processor encounters any errors while processing the SOAP message, e.g. while parsing the message the *Unprocessable* property is set to a boolean value *true*. This facilitates simple retrieval of unprocessable messages by administrators for debugging purposes and can be used to prevent repeated consumption of unprocessable messages.

4 Mapping WS interaction patterns to Linda coordination primitives

In the following paragraphs, the mapping of Web service *message exchange patterns* (MEP) to sequences of tuplespace operation calls is exemplified through the *In-out* MEP described as part of the WSDL 2.0 specification [6] and the custom *Request-for-bid* MEP that leverages the advanced functionality provided by the tuplespace transport.



(a) In-out MEP (b) Request-for-bid MEP

Figure 1. Web service MEPs mapped to Tuplespace operations

The *In-out* MEP as shown in Figure 1(a) comprises two message exchanges. First, the service requester sends a request message to the service provider. The service requester addresses the service provider either directly using the WS-Addressing *Destination* property which is propagated to the tuple level as the *Service name* property or indirectly using the *Action* property. This information is either extracted from the WSDL description of the Web service to be addressed or exchanged via out-of-band mechanisms and defined by the Web service requester. The request message is stored in the tuplespace by the service requester using the *out* operation and retrieved by the service provider using a blocking *in* operation. While certain tuplespace implementations such as JavaSpaces support asynchronous notifications as an alternative to blocking and non-blocking tuple consumption operations, a presentation of the message exchange patterns which makes use of these is left out for the sake of simplicity. The template used by the service provider to retrieve request messages matches on the *Service name* and the *Action* property of the request message. The destructive consumption operation (*in*) is used rather than its non-destructive variant (*rd*), since the message should be delivered to the service provider only once. When the request message has been processed by the service provider, it constructs a corresponding response or fault message by extracting the *Message ID* information (and the *Reply Endpoint* WS-Addressing header if found in the SOAP message encapsulated in the request tuple) from the request message. The *Correlation ID* property of the response message is set to the *Message ID* of the request message to relate the response message to the original request message and the *Relationship type* property is set to the URI representing a response message in a request-response interaction. The Web service provider stores the created response (or fault) message encapsulated in the content field of the response tuple in the tuplespace using the *out* operation. The service requester retrieves a response message for its original request message by issuing a blocking *in* operation, waiting for a message that contains the necessary correlation information that identifies the message as a response to the client's original request message.

The *Request-for-bid* MEP as shown in Figure 1(b) is an example for a complex Web service MEP that can be implemented effiently on top of a tuplespace. It is a composite pattern that consists of a *One-to-many* interaction and multiple *In-out* interactions. First, as part of the one-to-many interaction, potentially multiple service providers non-destructively consume a request message (*rd*); each service provider processes the request message and evaluates whether to send a corresponding response message to the service requester. If a service provider decides to respond to the request message, it acts as described in the *In-out* pattern. The MEP is terminated by the service requester by destructively retrieving (*in*) the request message from the tuplespace.

5 Conclusions

In this paper, we have motivated and presented a SOAP Web service binding for a tuplespace transport. We have payed special attention to pointing out which properties of tuplespaces motivate their use as a Web service transport. Furthermore with the example of the In-out and Request-for-bid MEPs, we have demonstrated the suitability of the presented transport to efficiently implement both the standard Web service message exchange patterns described in the WSDL 2.0 specification, as well as custom, more complex message exchange patterns. A more extensive description of the binding, further MEPs and a prototypical implementation of the proposed Web service tuplespace binding based on the *JavaSpaces* interface by SUN Microsystems is available in [8].

References

1. Gudgin, M., Hadley, M., Moreau, J.J.: SOAP Version 1.2 Part 1: Messaging Framework. W3C Recommendation 27 April 2007 (2007)
2. Gelernter, D.: Generative Communication in Linda. ACM Transactions on Programming Languages and Systems **7**(1) (1985) 80–112
3. Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., Berners-Lee, T.: RFC 2616: Hypertext Transfer Protocol HTTP/1.1. Jun **1** 999
4. Martin, D., Wutke, D., Scheibler, T., Leymann, F.: An EAI Pattern-Based Comparison of Spaces and Messaging. In proc. of EDOC 2007 (2007)
5. Freeman, E., Hupfer, S., Arnold, K.: JavaSpaces Principles, Patterns, and Practice. Pearson Education (1999)
6. Chinnici, R., Gudgin, M., Moreau, J.J., Schlimmer, J., Weerawarana, S.: Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language. W3C Working Draft **26** (2004)
7. Gudgin, M., Hadley, M., Rogers, T.: Web Services Addressing 1.0 - Core. W3C Recommendation (May 2006)
8. Schwind, A.: Space-Based Web Services: Konzepte und prototypische Implementierung mit Linda-Spaces. Master Thesis, DIP-2692, Universität Stuttgart, Fakultät Informatik, Elektrotechnik und Informationstechnik, Germany (Dec 2007)