Implementing Web Service Protocols in SOA: WS-Coordination and WS-BusinessActivity

Friedrich H. Vogt
Hamburg University of Technology

Simon Zambrovski
Hamburg University of Technology

Boris Gruschko
Hamburg University of Technology

Peter Furniss
Choreology Ltd.

Alastair Green
Choreology Ltd.

Abstract

Web Service protocol standards should be unambiguous and provide a complete description of the allowed behavior of the protocols' participants. Implementation of such protocols is an error-prone process, firstly because of the lack of precision and completeness of the standards, and secondly because of erroneous transformation of semantics from the specification to the final implementation. Applying the TLA+ paradigm we first consider the protocol on an abstract level. Safety properties taken from real world scenarios are compared to the facilities of the protocol. As result, we identified some limitation of applicability of the WS-BA protocol to abstract application use cases, modelled from the real world scenarios. These limitations are an omission of possible activities seen in the real world. Further, WS-C and WS-BA make assumptions about the internal structures of the participants, violating SOA paradigm. The former error could be detected by the use of formal methods. The latter can be circumvented by a sophisticated implementation strategy. The proposed strategy of implementing WS-Coordination and WS-BusinessActivity allows non-intrusive integration of the transactional framework, considering SOA requirements. This paper describes the results of analysis and some design decisions taken during the proof-of-concept implementation of WS-C and WS-BA frameworks.

1 Introduction

SOAP[6] based Web Services are seen as a solution for some interoperability problems of heterogenous distributed systems. This fact leads to rapid development of large number of protocols using SOAP conventions for message exchange. Transaction support is an important property of distributed systems. In the area of transactional Web Services, several protocols or groups of protocols have been proposed. One such set consists of the three protocols WS-Coordination[2], defining an extensible coordination framework, WS-AtomicTransactions[3], leveraging WS-Coordination(WS-C) for use with systems aware of ACID properties and finally, WS-BusinessActivities [4], designed for support of long-lived activities. The WS-C framework can be used with different coordination protocols, including the WS-AT and WS-BA specifications.

In this paper we describe our experience with implementing WS-C and WS-BA specifications and focus on guarding correctness of implementation. For this purpose we show how formal methods can help implementing safe systems. We also describe the decisions made during the design of proof-of-concept implementation and strategies adopted to deal with areas not precisely defined in specifications.

2 General approach

We separate the development process in four phases: the definition, the formal specification, the implementation and the validation.

First we choose the protocol stack according the application requirement. This requirement is stated in safety properties. Second we analyse given specifications. Ambiguity or lack of information found in specifications undermines the confidence level. To give a firm discussion base we use abstract models of the protocol behaviour. Those models are result of the specification phase. In case of WS-BusinessActivity and WS-Coordination the protocol state tables are not clear enough to serve as implementation model. Formal specifications are used to clarify issues in question. Using TLA+[10] we can check the consistency of such abstract models. This approach was inspired by formal modelling of WS-AtomicTransactions protocol described in[8]. Given the possibility to map the state transitions to exchanged messages, the generation of state graph of the system provides us with all possible sequences of messages

generated during an exchange. Because WS-BA only defines the behavior and message exchange of coordination protocols we are only interested in those messages.

For the implementation of the protocol we found the formal specification is a prerequisit for reaching the required level of confidence. In addition, it is more natural for the developer to handle the abstractions of TLA+ specifications than state transition tables. In [9] the consruction of a TLA+ specification for the WS-BA protocol is described as effortless, for an expert in the field of formal specifications. The time needed to construct the specification that checked safety properties, has been reported to be in the range of two hours.

To finally check the behavior of the resulting implementation we can use the validation approach proposed in [12] checking the message traces. To simplify the properties of traces to be checked the TLA+ specification can be used as an abstract input.

In following we give a short specification overview then discuss the ambiguities and areas of omission in the models, describe our proposals and finish the paper with validation approach followed by a conclusion.

3 Definitions

The WS-Coordination specification describes three roles for the communicating parties. The overview of the defined system model can be depicted from Fig. 1. An *Initiator* role is played by the entity aiming for a consensus among multiple Web Services. The *Participant* role is played by an entity offering some service that needs to be coordinated during the interaction. The *Coordinator* role is played by an entity coordinating the communicating parties to achieve the consensus. The specification also introduces the message exchange needed for the *Activation* and *Registration* of the participants. In the *Activation* phase, the `CoordinationContext` is acquired from the *Coordinator's* Activation Service. The `CoordinationContext`, a logical abstraction identifying the interaction is also defined in WS-C. It is attached to business messages being exchanged between the communicating parties, embedded in a SOAP header. In the *Registration* phase, the participant Web Service signals its interest on the mutual outcome of the coordinated interaction. During this phase the coordination protocol is negotiated and endpoint addresses of *Coordinator's* and *Participant's* protocol services are exchanged, forming a logical connection between *Coordinator* and *Participant*. The message flow over this logical connection depends on the coordination protocol being used and is not part of WS-C specification.

The WS-BusinessActivity specification defines two coordination protocols. These are the Business Activity With Coordinator Completion (BACC) as shown in Fig. 2 and Business Activity With Participant Completion (BAPC). BACC and BAPC are two-phase protocols, but differ from classic 2PC protocol [1] in the following manner. The first phase is used for exchange of business messages between the parties. In case of BAPC, the end of the first phase occurs when the `Completed` message is sent from *Participant* to *Coordinator*, indicating that the *Participant* has completed processing and stored all data persistently. The second phase is used for confirmation or negation of results achieved during the first phase.

The BAPC is designed for activities in which the decision about transition from the first to the second phase can be made by the *Participant*. The BACC is designed for activities in which this decision is made by *Coordinator*.

4 Specification Analysis

The work on the proof-of-concept WS-BA implementation was preceded by the reading and discussion of the specification itself. In this section we provide our insights and comments produced during the internal discussion of the specification useful for the understanding of the written

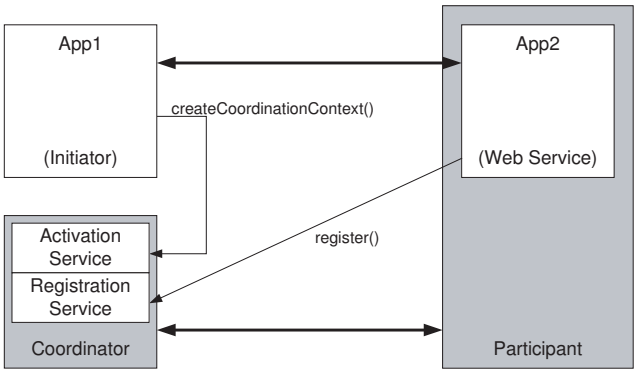

Figure 1. System overview (Specification)

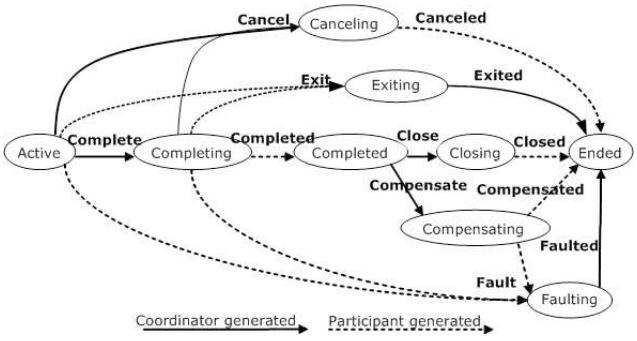

Figure 2. BACC

specification by a team about to implement it.

4.1 Application models

Concerning on formal analysis of the WS-C and WS-BA specifications using TLA+ paradigm led to review of applicability of the protocols to the business scenarios taken from the real world. We came to a conclusion that WS-BA can only support ”do-compensate”[7] behavior patterns for the *Participants*. In the ”do-compensate” model the confirmed state is identical to the provisional state. In other patterns such as ”provisional-final” and ”validate-do” behavior patterns[7], the participant establishes a (provisional) ”complete” state of the application data that will be changed to the confirmed state if and when the `Close` message is received. The fact that in WS-BA the `Closing` state has no state transition to the `Faulting` state means that no action on data can be performed while the system is in that state. WS-BA permits a transition to the `Faulting` state from the `Compensating` state since it recognises that any change of application state can sometimes turn out to be impossible. Supporting the other patterns by adding the `Closing` to `Faulting` state transition would result in a wider applicability of WS-BA to natural scenarios, especially those where the release of application data in its final state will have wider consequences. In terms of service-oriented design the behavior supported by WS-BA coordination protocols violate the SOA paradigm, prescribing the internal behavior of the system. This prescription results from the negotiation of the coordination protocol, where the participant commits himself to an internal behavior pattern supported by the negotiated protocol.

4.2 Analysis Results

WS-Coordination and WS-BusinessActivity are protocol frameworks designed for usage in the SOA environment. Nevertheless the protocol authors made some decisions about the internal buildup of communication parties as described in [5]. The tightly-coupled *Coordinator* and *Initiator* as well as *Participant* encapsulating both business and transactional logic are examples of that. Our understanding of WS-* protocols as building blocks of distributed system led to different view of the system than described in [5]. Specifically our architecture was shaped by consideration of seamless integration of WS-C and WS-BA frameworks in existing WS-Scenarios minimizing the adaptation efforts. For this purpose we introduce the transactional middleware separating the coordination from the business logic as shown in Fig. 3. The function of the transactional middleware is the management of the coordination context and coordination protocol execution. By assuming this assignment, the transactional middleware allows for an easier *Client* implementation. (The addition of the transactional middleware brings our WS-BA implementation closer to the architecture described in WS-AtomicTransactions[3], where the WS-AT Completion protocol is analogous to the message exchanges between the client and the middleware described below.)

4.2.1 Initiation and Termination

The WS-C specification defines a message flow that has to be understood by all communication parties. To allow non-intrusive integration of WS-C framework with existing Web Services and their clients we introduce mechanisms for enabling and disabling WS-Coordination support on demand. For this purpose the model defined in WS-C specification is extended with a new role called *Transactor*. The *Transactor* accepts four different messages from *Initiator* that deal with initiation and termination of coordination support, as well as lead to the final coordinated outcome of the protocol in use. Transactional support for interactions between Web Services and their clients begins when the *Initiator* sends a `Begin` message to *Transactor*. Similarly, to end the transactional support the `End` message is sent to *Transactor*. The *Transactor* form the first part of transactional middleware as seen in Fig. 3.

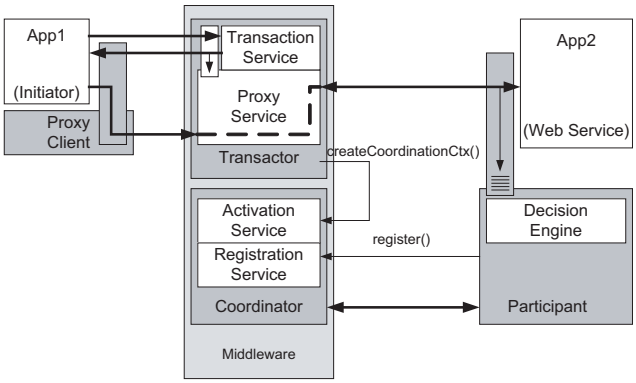

Figure 3. System overview(Implementation)

4.2.2 Delivery of decisions

In both WS-BusinessActivity protocols the participant reaches the `Completed` protocol state. In the BAPC protocol, the *Participant* reaches this state after it has sent the *Coordinator* a `Completed` message. According to the BACC (see Fig. 2) protocol, the *Participant* reaches this state after receiving the `Complete` message from the *Coordinator*, and having successfully progressed through the `Completing` state. In the `Completed` state the logic on the participant side has recorded all its business data and

expects a decision from the *Coordinator* about further protocol progression, which should eventually lead to protocol instance termination. In general, however, the *Coordinator* has no ability to understand the semantics of the business messages being exchanged between the client and the Web Service. In particular it has no knowledge about the business process flow. This knowledge is only available to the client acting as *Initiator*. This means the *Coordinator* cannot decide by itself which message to send to the *Participant* after the *Participant* has reached the `Completed` state. For this purpose we extend the *Transactor* by introducing the ability to transmit a decision of the client to the *Coordinator*. This decision is business flow dependent and enables the *Coordinator* to send the appropriate coordination protocol message to the *Participant*. For this to happen, the *Transactor* can receive two messages from the *Initiator*, namely a `Confirm` message, and a `Cancel` message. This decision indication received by the *Transactor* is made available to the tightly-coupled *Coordinator*. Also important to mention here is that these `Confirm` and `Cancel` messages include the endpoint address of the Web Service (to which the earlier business messages were sent) so that the decision by the client can be associated with the particular Web Service. The transactional middleware consists of the *Transactor* and the *Coordinator*, which is thus decoupled from the *Initiator*.

4.3 Specification Omissions

4.3.1 Registration

The WS-Coordination specification prescribes that the *Participant* register with the *Coordinator* if it intends to participate in a coordinated business activity. However, the `Register` message does not contain enough information for the *Coordinator* to determine which business activity the *Participant* wants to take part in. It is possible to resolve this lack of information in several ways. For example, the *Coordinator* could provide distinct *Registration Service* endpoint addresses for each business activity. We chose another approach and extended the `Register` message interaction by adding the missing information in the form of identification information of the *Participant*. We also provide the address of the business Web Service endpoint in the `Register` message. Given the possibility of a *Participant* taking part in several business activities simultaneously, our extension of the `Register response` message allows its assignment to the corresponding business activity. The `CoordinationContextIdentifier`, defined in WS-C specification for identifying the coordinated interaction uniquely, has been used as the extension for both messages. An example emphRegister message with the identifier is shown in Listing 1 in which the `CoordinationContextIdentifier` is provided in `wsu:Identifier` element.

```
<wscoor:Register
  xmlns:wscoor="..." xmlns:wsa="..."
      xmlns:wsu="..."
>
 <wscoor:ProtocolIdentifier>
     http://schemas.xmlsoap.org/ws/2004/01
     /wsba/
     BusinessAgreement
     WithParticipantCompletion
 </wscoor:ProtocolIdentifier>
 <wscoor:RequesterReference>
   <wsa:Address>
       http://example.org/Request
   </wsa:Address>
 </wscoor:RequesterReference>
 <wscoor:ParticipantProtocolService>
   <wsa:Address>
       http://example.org/BAPCPort
   </wsa:Address>
 </wscoor:ParticipantProtocolService>
 <wsu:Identifier>
       http://example.org/?id=1
 </wsu:Identifier>
 <wsa:EndpointReference>
   <wsa:Address>
       http://example.org/BusinessPort
   </wsa:Address>
 </wsa:EndpointReference>
</wscoor:Register>
```

Listing 1. Register Message

4.3.2 Coordination Protocols Extension

Both the *Coordinator* and the *Participant* can hold several coordination protocol instances simultaneously. The WS-BusinessActivity specification does not provide enough information to differentiate between coordination protocol instances. Similar to the case of `Register` and `Register response` messages we include the identification element in the messages to allow the receiving party unique assignment between the coordination messages and corresponding protocol instances.

5 Implementation

The separation of *Coordinator* from *Initiator* has been enabled by usage of a *Proxy System*. The *Proxy System* consists of two parts: a *Proxy Client* deployed on the *Initiator* side and *Proxy Service* as a part of proposed transactional middleware. The *Proxy Client* is realised as a SOAP handler intercepting the messages, redirecting them to the *Proxy*

Service, which is a part of *Transactor*. The initial creation of `CoordinationContext` is ensured on the middleware, which augments the rerouted business messages with `CoordinationContext` of corresponding business activity.

Our definition of the *Participant* differs slightly from that described in WS-C specification. We describe only the transactional component of the business Web Service as the *Participant*. Since the *Participant* and the business Web Service have different roles, the former being responsible for coordination, and the latter for business functionality, it is good design to keep them separate. On the other hand, coupling between the two roles is required for mutual exchange of information about their internal states, since proper progression of coordination depends on these states. There are several approaches for a business Web Service to inform the *Participant*, or for the *Participant* to inform the business Web Service about the changes of their respective internal states. Our approach of loose-coupling of the business Web Service and the *Participant* is based solely on the observation of the in- and outbound communication of the business Web Service. Using *Decision engine* linked with a SOAP handler intercepting the messages of the business Web Service the *Participant* concludes the change of internal state of the business Web Service. For this purpose the *Decision Engine* is equipped with a preconfigured *Rule Set* constisting of XQuery[13] predicates. The recorded messages are written into *Trace*[12] data structure, which is used as container for *Rule Set* evaluation. Further discussion of the applicability of the proposed approach and the *Rule Set* is beyond the scope of this paper and is a subject of further research. The concept of *Decision Engine* minimizes the effort needed to adapt an existing business Web Service for usage with WS-BA to writing of a configuration file containing the mappings between the coordination and business expressions. For simulation purposes a common travel agency scenario has been implemented. A complete example interaction depicting the components described previously is shown in Fig. 4.

We packaged our WS-BA framework implementation as an J2EE application, that has been deployed in two JBoss Application Servers. Apache Axis 1.2 has been used as Web Service toolkit. For the usage of TLA+ language an Eclipse IDE plugin has been developed[14].

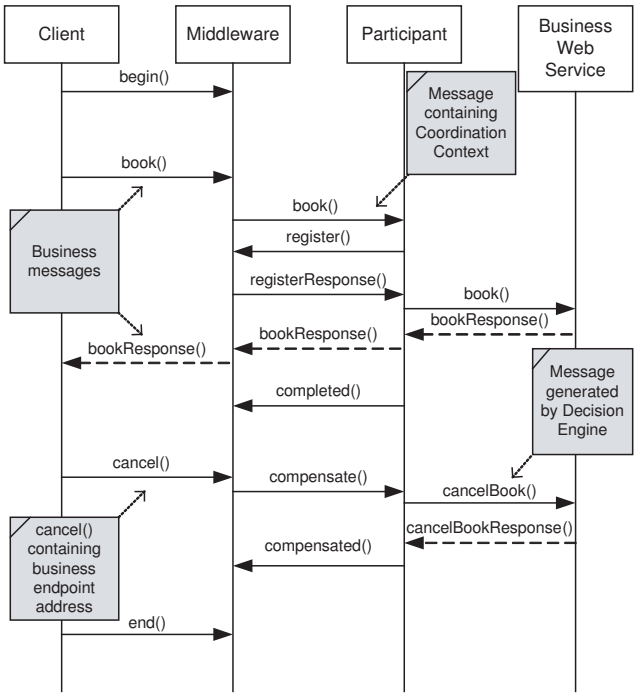

Figure 4. Sample interaction scenario

6 Validation of the Implementation

Validation of the implementation of a given protocol provides the confidence in the correctness of decisions met during the implementation process. It is hard to verify an implementation of the presented system. The mathematical validation of the implementation seems unfeasible, due to the complexity of the matter at hand. Nevertheless, we show a way to acquire an assertion about the correctness of the implementation. To test our implementation we used the method suggested in [12]. The proposed approach allows separation of the implementation details from the testing of the overall compliance with the WS-BA specification. The *Traces* which are used for the *Decision Engine* are at the same time being stored for later evaluation.

As proposed in [12] the evaluation of the *Traces* does not prove the definitive compliance of the implementation with the WS-BA specification. It merely guarantees that no specification violation has been observed. The validation relies upon a set of predicates which provide a description of the constraints laid upon the communication by the WS-BA specification.

The main effort during the proposed validation is needed to be applied to the creation of the predicates set. This set is specific to the protocol which implementation is to be tested. Thus there is no possibility to reuse a created predicates set for testing the implementation of another protocol. A useful base for the creation of the predicates set is a formal specification of the state transitions of the protocol. From this specification a comprehensive set of predicates can be derived. Another advantage of using a formal specification is the avoidance of errors in the predicates set.

7 Conclusion

The formal analysis of the WS-Coordination and WS-BusinessActivity specifications led to determination of ambigous areas in the described frameworks. The TLA+ paradigm helped us perform this analysis. During the analysis phase we uncovered a limitation of the specifications in terms of applicability to real world scenarios. In our understanding of SOA this limitation violates the black box approach to the behaviour of the participants. We accepted this limitation for the cause of overall interoperability of our implementation. Further we discovered a structural dependancy between introduced entities. This also violates the SOA paradigm. This dependancy could be resolved by sophisticated design of the WS-BA framework implementation. The introduction of transactional middleware forms a loosly-coupled transactional system according to WS-C and WS-BA specifications. To allow for the mapping between incoming messages and their corresponding BAs we took advantage of the extensibility of elements descibed in WS-C and WS-BA specifications. The easy integration into existing Web Service scenarios is enabled by the usage of *Proxy System* and *Decision Engine*, whose functionality is described in the Sec. 5 The communication protocol defined between the *Initiator* and *Transactor* is needed to guarantee the loose-coupling of system components. The insights gained during the proof-of-concept implementation emphasize our analysis and design decisions. The proof-of-concept implementation has been exposed to an extended validation phase using data gathered during the test runs of an example scenario. The overall experience shows, that the usage of formal methods during an implementation of Web Service protocols in SOA helps clarify the protocols under consideration and raises the confidence of the implementors into their understanding of the protocols.

References

[1] P.A. Bernstein and E. Newcomer. *Principles of Transaction Processing*. Morgan Kaufmann Publishers, 1997.

[2] Luis Felipe Cabrera, George Copeland, William Cox, Max Feingold, Tom Freund, Jim Johnson, Chris Kaler, Johannes Klein, David Langworthy, Anthony Nadalin, David Orchard, Ian Robinson, John Shewchuk, Tony Storey, and Satish Thatte. Web Services Coordination Framework (WS-Coordination), September 2003.

[3] Luis Felipe Cabrera, George Copeland, William Cox, Tom Freund, Johannes Klein, David Langworthy, Ian Robinson, Tony Storey, and Satish Thatte. Web Services Atomic Transaction Framework(WS-AtomicTransaction)), Januar 2004.

[4] Luis Felipe Cabrera, George Copeland, William Cox, Tom Freund, Johannes Klein, David Langworthy, Ian Robinson, Tony Storey, and Satish Thatte. Web Services Business Activity Framework (WS-BusinessActivity), Januar 2004.

[5] Luis Felipe Cabrera, George Copeland, Jim Johnson, and David Langworthy. Coordinating Web Services Activities with WS-Coordination, WS-AtomicTransaction, and WS-BusinessActivity. Available at http://msdn.microsoft.com/webservices/default.aspx, January 2004.

[6] R. Chinnici, M. Gudgin, J.-J. Moreau, and S. Weerawarana. SOAP Services Description Language (WSDL) 1.2, March 2003. status : W3C Working Draft , http://www.w3.org/TR/wsdl12/.

[7] Peter Furnis and Alastair Green. Choreology Ltd. Feedback to the authors of WS-Coordination, WS-AtomicTransaction and WS-BusinessActivity. Available at http://www.choreology.com/downloads/, May 2004.

[8] James E. Johnson, David E. Langworthy, Leslie Lamport, and Friedrich H. Vogt. Formal specification of a web services protocol. *Electronic Notes in Theoretical Computer Science, Volume 105, 10 December 2004, Pages 147-158*, December 2004.

[9] James E. Johnson, David E. Langworthy, Leslie Lamport, and Friedrich H. Vogt. Formal specification of a web services protocol. *to appear in Elseview Science*, January 2005.

[10] Leslie Lamport. *Specifying Systems*. Addison Wesley, 2003.

[11] Eric Newcomer and Greg Lomow. *Understanding SOA with Web Services*. Addison Wesley Professional, 2004.

[12] Marcus Venzke. Specifications using XQuery Expressions on Traces. *Mario Bravetti, Gianluigi Zavattaro (Eds.): Proceedings of the First International Workshop on Web Services and Formal Methods*, February 2004.

[13] W3C. XQuery: the W3C query language for XML – W3C working draft. Available at http://www.w3.org/TR/xquery/, 2001.

[14] Simon Zambrovski and Boris Gruschko. TLA+ Eclipse IDE Plugin. Available at http://www.techjava.de/, 2004.