Design choices and strategies for implementing WS-BusinessActivity specification. *

Simon Zambrovski
Technical University
Hamburg-Harburg
Schwarzenbergstrasse 95
21073 Hamburg, Germany
simon@zambrovski.org

Boris Gruschko
Technical University
Hamburg-Harburg
Schwarzenbergstrasse 95
21073 Hamburg, Germany
boris@gruschko.org

Muhammad F. Kaleem
Technical University
Hamburg-Harburg
Schwarzenbergstrasse 95
21073 Hamburg, Germany
m.kaleem@tu-harburg.de

ABSTRACT

This paper describes an implementation of the Web Services Business Activity Framework (WS-BusinessActivity), which is an industry specification aimed at providing coordination protocols for coordinating long-running transactional activities in a Web Services environment. This specification itself leverages the extensible coordination framework described in the Web Services Coordination specification. These specifications provide a general overview of the frameworks and define the requisite interfaces, however no design and implementation issues are described in them. In this paper we introduce some concepts and strategies for implementing the frameworks described in these specifications, and focus especially on the WS-BusinessActivity specification. We also focus on how existing Web Services may leverage the transactional functionalities offered by these frameworks without requiring any change to their existing interfaces. We describe the design of our implementation and highlight the strategic choices we made, specially about aspects not explicit in the WS-BusinessActivity specification.

Categories and Subject Descriptors

H.3.4 [**Information storage and retrieval**]: Systems and Software—*Distributed systems*; H.3.5 [**Information storage and retrieval**]: On-line Information Services—*Web-based services*

Keywords

Distributed transaction systems, WS-BusinessActivity, distributed transactions, Web Services

*A full version of this paper is available in german as a report *Entwurfskonzepte und Implementierungsstrategien für das WS-BusinessActivity Frameworks* at http://jwst.sf.net/resources/publications/wsba-report.pdf

ICSOC04 2004, 2nd International Conference on Service Oriented Computing, New York City, NY USA

1. INTRODUCTION

Support for transactions is an important property of distributed systems. Traditional transactional models usually involve an entity such as a transaction manager (or a coordinator) having strict control over the transaction participants. Such a model is not directly applicable to a Web Services environment, which consists of autonomous services spread over the internet. Service providers cannot be expected to open up their services to direct control or management by external entities. However, it may be desirable to have autonomous Web Services cooperate within the scope of a transaction. For such situations, extended transactional models have been designed that cater to requirements of a Web Services environment. One such model consists of the Web Services Coordination (WS-C)[1] and Web Services BusinessActivity (WS-BA)[2] specifications. The WS-C specification defines a coordination framework for Web Services, and the WS-BA specification leverages this coordination framework to define coordination protocols suitable for coordinating long-running activites based on Web Services interactions.

1.1 The specifications

WS-C Specification implies three roles for the communication parties. The *Initiator* is an entity interested in a coordinated interaction with multiple Web Services. Typically it would be a client application invoking different Web Services in the scope of an activity. The *Participant* is an entity offering some business service that needs to be coordinated during the interaction. The *Coordinator* is an entity coordinating the interaction. It also offers an Activation Service and a Registration Service. This is represented in Figure 1. The message flow between the different entities is as follows. Prior to the begin of the communication the Initiator retrieves a *Coordination Context* from the Coordinator's Activation Service. The Coordination Context is a logical abstraction identificating the data exchange for a particular activity. The *Coordination Context Identifier* uniquely identifies a Coordination Context. The Initiator begins its communication with Participants by propagating the Coordination Context within the business messages. The receiving of a message containing the Coordination Context by the Participant allows the assignement of the message to particular coordinated activity. If such message is received for the first time, the Participant registers itself for the coordinated activity by the Coordinator's Registration Service. During

this registration the *Coordination Protocol* is negotiated and the addresses of corresponding *Coordination Protocol Endpoint* are exchanged.
Different coordination scenarios have different coordination requirements. This need is reflected in the WS-C specification by support of pluggable coordination protocols. WS-BA specification defines coordination protocols for long-running transactions, and names such a transactional activity as *Business Activity*(BA). A Business Activity can be partitioned in scopes. For simplicity we omit the support of scopes in this paper, because this feature is only relevant for the Coordinator's internal logic and can be seamlessly integrated into the implementation on demand.

1.2 The coordination protocols

WS-BusinessActivity specification defines two coordination protocols. These are *Business Activity With Coordinator Completion* (BACC) protocol and *Business Activity With Participant Completion* (BAPC) protocol. These protocols allow the Coordinator to transmit some coordination information to the Participants. BACC and BAPC are two-phase protocols, but differ from classic two-phase-commit protocol[7] in the following manner. During the first phase some business data is exchanged. The end of the first phase is marked with *completed*-message from the Participant, which indicates that all data handled by the Participant is stored persistently. The second phase is used for confirmation or negotiation of results achieved during the first phase.
The BAPC protocol should be used for activities in which the Participant can make a decision about the transition from the first to the second phase. The BACC protocol should be used for activities, in which this decision is made by the Coordinator.

2. SPECIFICATION ANALYSIS

In this section we analyze the design and architectural requirements of the WS-C and WS-BA frameworks, in search of constraints for the system to be built. We also highlight some ambigous areas of these specifications and provide concrete suggestions for them.

2.1 Inititation and Termination

WS-Coordination defines a message flow that should be understood by all communicating parties. To allow stepless integration of WS-Coordination framework within existing Web Services and thier clients we introduce mechanisms for activation and deactivation of WS-Coordination support on demand. For this purpose we introduce a new role into the interaction model and name it *Transactor*. The two operations offered by Transactor are important for the initiatiation and termination of transactional communication. The *begin* message activates the support and the *end* message deactivates it.

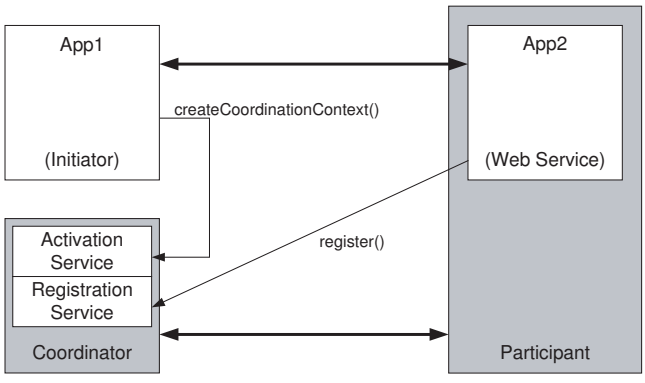

Figure 1: System overview(Specification)

2.2 Registration

The WS-C specification prescribes the Participant to register with the Coordinator if it intends to participate in a business activity. 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. The Coordinator could provide distinct Registration Service endpoints for each BA. We chose another approach and extended the *register* message by the missing information. We also provide the address of the business endpoint in the register message. Due to the possibility of a participant taking part in several BAs simultaneously, our extension of the *register response* message provides identification information for the Participant to assign it to the corresponding BA. The Coordination Context Identifier has been used as the extension for both messages. An example register message with the identifier is shown in listing 1.

```
...
<wscoor:Register xmlns:wscoor="..."
    xmlns:wsa="..." xmlns:wsu="...">
    <wscoor:ProtocolIdentifier>
        http://schemas.xmlsoap.org/ws/2004/01/
        wsba/BusinessAgreementWith
        ParticipantCompletion
    </wscoor:ProtocolIdentifier>
    <wscoor:RequesterReference>
        <wsa:Address>
            http://example.org/RegistrationReq
        </wsa:Address>
    </wscoor:RequesterReference>
    <wscoor:ParticipantProtocolService>
        <wsa:Address>
            http://example.org/BAPCProt
        </wsa:Address>
    </wscoor:ParticipantProtocolService>
    <wsu:Identifier>
        http://example.org/?identifier=1
    </wsu:Identifier>
    <wsa:EndpointReference>
        <wsa:Address>http://example.org/
            BusinessPort</wsa:Address>
    </wsa:EndpointReference>
</wscoor:Register>
...
```

Listing 1: Register Message

2.3 Delivery of decisions

Both WS-BusinessActivity protocols contain the *completed* protocol state. In this state the protocol logic on the Participant side has recorded all business data and expects a decision from the Coordinator about further protocol progression. There are two possibilities: the *compensate* message serves to undo the work of the participant, the *close*

message confirms the performed work.
In general the coordinator has no ability to understand the semantics of the business messages being exchanged between the client and the Web Service. Particularly it has no knowledge about the business process flow. This knowledge is only available on the client side. This means it cannot decide by itself which message to send to the Participant.
For this purpose we extend the Transactor service by introducing the ability to transmit a decision of the client. This decision is business flow dependent and enables the Coordinator to send the appropriate message to the Participant. Message containing this decision also includes the Business Endpoint Address of the Web Service to associate the decision with a particular Web Service.

2.4 Coordination protocols extension

Both the Coordinator and the Participant can hold several coordination protocol instances simultaneously. The WS-BA 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.

3. USED MODELS

The WS-C and WS-BA specifications combined with extension we described in the previous section define four different entities communicating with each other. The important aspect of our implementation strategies are the interfaces between the existing communication parties and the introduced entities. In this section we introduce some conceptual models for development and integration of the components. In doing so, we focus on achieving maximum loose-coupling of components.

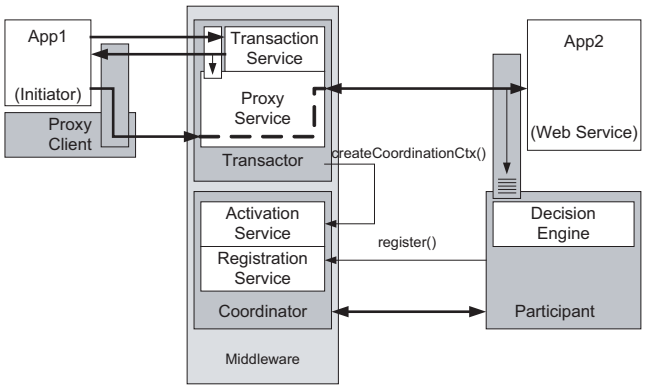

Figure 2: System overview(Implementation)

3.1 Message interception

The basic mechanism for extension of messages defined in SOAP[4] is support of headers. It allows to provide additional metadata within a message. This extension point is used in message flow defined by WS-C by putting coordination context in each message exchanged between the client and the Web Service. This mechanism becomes more efficient if communicating parties do not need to handle headers. For this purpose we introduce a *Message Interceptor*. Message Interceptor is a software component, that is specific to the Web Service Toolkit vendor. It is based on Interceptor J2EE Design Pattern. The goal of this component is to read the coordination SOAP-headers from incoming messages and to write coordination SOAP-headers to the outgoing messages. The usage of Message Interceptor simplifies the structure of the communicating parties and separates the transaction system from the business service.

3.2 Proxy

The description of WS-C and WS-BA frameworks in [3] is based on the assumption, that the Initiator is tightly coupled with Coordinator or acts as as the Coordinator itself. Furthermore, the Initiator is the client application that accesses the Web Service. This assumption certainly holds for analysis of coordinated communication but leads to dependency between the business and transaction systems during the implementation. To minimize this dependency we introduce a *Proxy System*. Its main idea is to route the business message flow between the client and the Web Service to the Middleware system as shown in Figure 2. The relocation of Initiator and Coordinator entities to this Middleware is then possible. The Proxy System consists of two parts a *Proxy Client* and a *Proxy Service*.

3.2.1 Proxy Client

A Proxy Client is a Message Interceptor deployed on the message path in the client application. It is inactive until the client sends the *begin* message to the Transactor. If active, the Proxy Client sends all intercepted messages to the Proxy Service. The deactivation of the Proxy Client is triggered on the receipt of *end response* message from the Transactor.

3.2.2 Proxy Service

The Proxy Service is a Web Service tightly coupled with Transactor, using a Message Interceptor installed on its message path. The main task of Proxy Service is to reroute the messages to their destinations aimed at client application. The response messages received from the Web Services are then rerouted to the client. All messages exchanged between the Proxy System components contain the Proxy-SOAP-Header as shown in listing 2, which identifies the *Proxy Connection*.

```
...
<soapenv:Header>
   <prx:ProxyReference xmlns:prx="..."
       xmlns:wsa="...">
       <prx:EndpointReference>
           <wsa:Address>
               http://example.org/services/
                   ProxyService
           </wsa:Address>
       </prx:EndpointReference>
       <prx:ProxyID>17</prx:ProxyID>
   </prx:ProxyReference>
</soapenv:Header>
...
```

Listing 2: Proxy Response Header

3.3 Decision engine

Similar to the dependency between the client and the Coordinator and Initiator the dependency between the business Web Service and Participant is assumed in [3].

There are several approaches for a Web Service to inform the Participant, or for the Participant to inform the Web Service about the changes of their respective internal states. The Web Service could offer an additional API to support a mutual exchange of internal state information with the Participant. The drawback of this approach is the changes required in the Web Service's API. We propose another approach to resolve the dependency between the Web Service and the Participant. Our approach is based solely on the observation of the in- and outbound communication of the Web Service. A Message Interceptor is installed on the path of the in- and outbound messages of the Web Service. The intercepted messages are written into a Trace[8] data structure. An example of a Trace is given in listing 3. Upon

```
<trace:TraceType xmlns:trace="...">
   <trace:Message>
      <soapenv:Envelope
         xmlns:soapenv="...">
         ...
      </soapenv:Envelope>
      <trace:operation/>
      <trace:to>Client</trace:to>
   </trace:Message>
</trace:TraceType>
```

Listing 3: Trace

arrival of a Message a *Decision Engine* software component is triggered. The Decision Engine determines if the Web Service state transition has occurred. To allow such conclusions the Decision Engine is preconfigured with a Web Service specific Rule Set. The Rule Set consists of XQuery[9] predicates which will be evaluated on the Trace. The result of the evaluation enables the Decision Engine to conclude the occurrence of a change in the Web Service's internal state. The RuleSet provides a mapping between the internal state of the Web Service and the changes of the state of the Coordination Protocol in Participant. As soon as the Coordination Protocol Instance on the Participant side reaches the *completed* state, it waits for decision from Coordinator side about further protocol progression. As described above the usage of Transactor allows the transmission of this decision from client to Coordinator, which retransmits it to the Participant. Using the Decision Engine with another pre-configured set of expressions it is possible to construct a SOAP-message for sending to business Web Service. This SOAP-message contains the decision of the client transformed into the corresponding message from the business domain of Web Service.

The discussion of the applicability of the proposed approach and the RuleSet 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. A

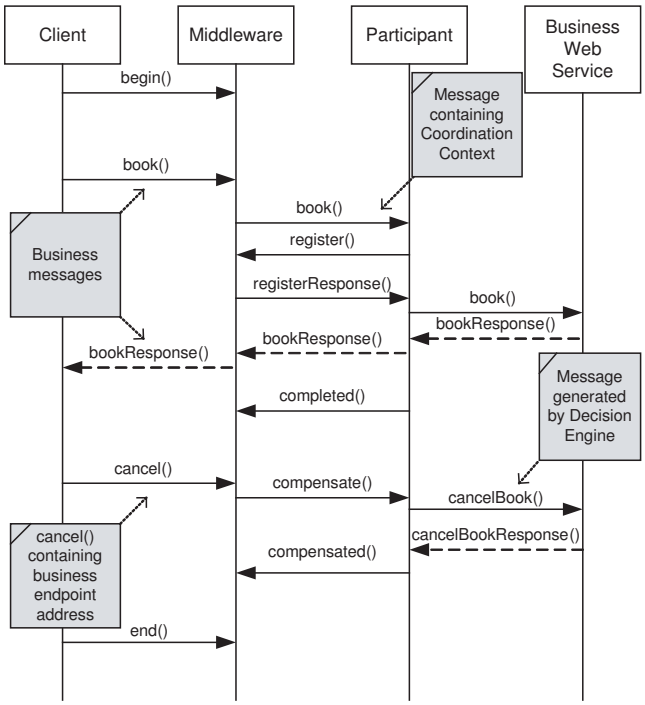

Figure 3: Sample interaction scenario

complete example interaction depicting the components described previously is shown in figure 3.

4. IMPLEMENTATION DETAILS

In the previous section we introduced design concepts needed for the implementation. In the following section we give some details about our implementation.

4.1 Implementation platform

The WS-BA Framework has been implemented in Java, using J2SDK 1.4 from Sun Microsystems. On the server side we used JBoss Application Server 3.2.3 as J2EE Web Container and JBoss JMS and JBoss JMX. For the Web Services used in the framework Apache AXIS 1.2 has been used as a Web Service toolkit. It has been rebuilt with Castor 0.9 to allow better serializer/deserializer support. A sample travel agency scenario has been developed in order to validate the functionality of our implementation. The scenario implementation uses JBoss J2EE EJB container with MySQL 4.0 as data storage.

4.2 Web Service implementation

Web Services introduced in the previous section has been implemented using Apache AXIS. The XML Schema documents used in WS-C and WS-BA specifications has been mapped to corresponding Java objects with Castor[6]. As described in subsection 3.1 some Web Services use Message Interceptors installed on the message path. Those Message Interceptors has been implemented as AXIS handler. The configuration of the different message interceptors has been performed using standard parameter bag supported by AXIS deployment descriptors.

4.3 Decision Engine Implementation

The execution of the decision engine predicates on the Web Service Traces has been implemented using the Saxon XQuery Processor. For performance reasons the recording of the Message Trace has decoupled from the message flow via a JMS Queue.

4.4 Monitoring

We implemented a JMX interface for our implementation to enable the external monitoring of system's internal state. The exposed information contains the Coordination Protocol states of the Coordination Protocol Instances. To visualize the changes of the Coordination Protocol states we developed a graphical JMX client which is depicted in Figure 4.

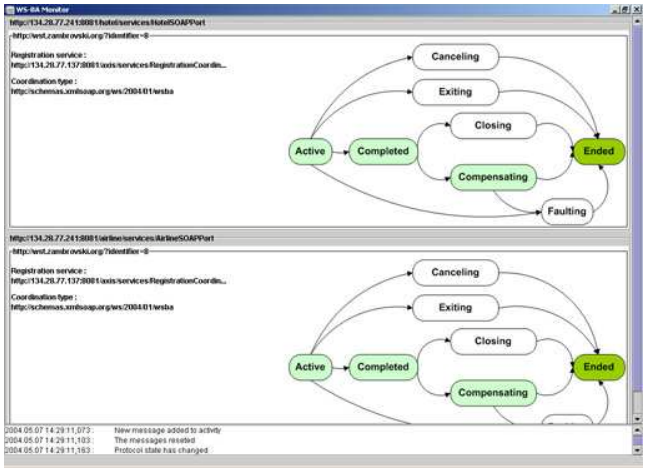

Figure 4: Monitoring Client

5. CONCLUSION

The proposed approach separates the business from the transaction handling Web Services. This separation is enabled by the Message Interceptor installed on the message path. On the client side the Message Interceptor allows placement of the Initiator in the Middleware. This simplifies the implementation of the client. The Proxy System approach defines a scope on messages being exchanged during the transactional communication. The messages demarcated by this scope will be delivered to the Middleware and then rerouted to the business Web Service. The Coordination Context is assigned by the Middleware to each message which has been sent in the scope of a transactional communication. The usage of Transactor enables the placement of the Coordinator in the Middleware. The Decision Engine allows an external determination of the internal state of the Web Service. This capability allows retention of the interface of a Web Service, while enabling it's participation in a Business Activity. In order to allow further evaluation in educational and research domain we published our implementation on the SourceForge website. It can be reached via it's project site at http://sf.net/projects/jwst/.

6. REFERENCES

[1] L. F. Cabrera, G. Copeland, W. Cox, M. Feingold, T. Freund, J. Johnson, C. Kaler, J. Klein, D. Langworthy, A. Nadalin, D. Orchard, I. Robinson, J. Shewchuk, T. Storey, and S. Thatte. Web Services Coordination Framework (WS-Coordination), September 2003.

[2] L. F. Cabrera, G. Copeland, W. Cox, T. Freund, J. Klein, D. Langworthy, I. Robinson, T. Storey, and S. Thatte. Web Services Business Activity Framework (WS-BusinessActivity), Januar 2004.

[3] L. F. Cabrera, G. Copeland, J. Johnson, and D. Langworthy. Coordinating Web Services Activities with WS-Coordination, WS-AtomicTransaction, and WS-BusinessActivity, 2004.

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

[5] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Massachusetts, 1994.

[6] K. Gibbs, B. D. Goodman, and E. Torres. Create Web services using Apache Axis and Castor. Available at http://www-106.ibm.com/developerworks/webservices/library/ws-castor/, 2003.

[7] J. Gray and A. Reuter. *Transaction Processing*. Morgan Kaufmann Publishers, San Mateo (CA), USA, 1993.

[8] M. 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.

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