Web Services Conversation Language (WSCL) 1.0

May 2001

Authors (alphabetically): Arindam Banerji, Claudio Bartolini, Dorothea Beringer, Venkatesh Chopella, Kannan Govindarajan, Alan Karp, Harumi Kuno, Mike Lemon, Gregory Pogossiants, Shamik Sharma, Scott Williams

Abstract

This document specifies the Web Services Conversation Language WSCL. WSCL allows defining the abstract interfaces of web services, i.e. the business level conversations or public processes supported by a web service. WSCL specifies the XML documents being exchanged, and the allowed sequencing of these document exchanges. WSCL conversation definitions are themselves XML documents, and can therefore be interpreted by web services infrastructures and development tools. WSCL may be used in conjunction with other service description languages like WSDL, e.g. to provide protocol binding information for abstract interfaces and to specify the abstract interfaces a concrete service is supporting.

Status, Contact

This draft proposes a simple conversation language standard that can be used for various web-service protocols and frameworks, and may but needs not be combined with WSDL. As such it fills the gap between mere interface definition languages that do not specify any choreography, and more complex process or flow languages that describe complex global multiparty processes. Feedback and comments are welcome, send them to dorothea_beringer@hp.com.

Contents

Introduction……. 1
Elements of a WSCL Specification …… 4
Mapping to Protocol Bindings and Error Handling …….. 12
Using WSCL and WSDL to Describe Web Services …… 13
Extending WSCL …….. 15
References …….. 15
Appendix A: WSCL Schema Definition ……. 17
Appendix B: Example WSCL Specification …….. 19

Introduction

Electronic commerce is moving towards a vision in which corporate enterprises use web services to interact with each other based on well-defined standards in a dynamic and loosely coupled way. In order for effectively using web services, the interacting parties need to know and agree upon the following three aspects:

- Business payload: Both parties need to know which information to exchange.
- Protocol: Both parties need to know how to exchange business payload, i.e. they have to agree on the message structure, message header information, mapping to

underlying transfer protocols, and the overall framework for communication and error handling. Protocols for web services are e.g. SOAP-RPC [2], asynchronous SOAP with specific profiles, RNIF [3], ebXML TR&P [4], etc.

- Service location: In order to interact with a specific service, we need to know which protocols it supports, which payload it exchanges and also its location, e.g. an HTTP URL.

In the context of web services, business payload is mostly expressed in XML. Therefore, one possible and often used way to define the business payload is to use XML schemas. Yet defining which XML documents are expected by a web service or are sent back as a response is not enough. We also have to define in which order these documents need to be exchanged, thus specifying a business level conversation. By specifying the conversations supported by a web service, i.e. by defining the documents to be exchanged and the order in which they may be exchanged, we define the external visible behavior of a web service, its abstract interface. Conversations focus on the public processes in which the participants of a web service engage. Conversations may not define the application logic or private process, i.e. the internal implementation and mapping to back-end applications within the various enterprises that are interacting with each other.

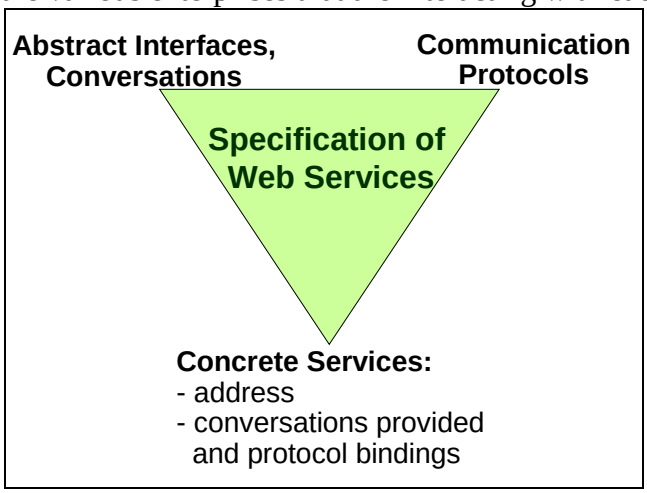

Figure 1: Basic concepts for specifying web services

By clearly distinguishing between the concepts of abstract interface, communication protocol, and concrete service, we enable the loose coupling desired for web services. All three aspects can be specified independent of each other. Protocols can be specified once for very different web services. The abstract interfaces defining the business-level conversations can be specified once for a specific industry and then be bound to different protocols and be used by any number of services with completely different implementations. A single concrete service can implement several abstract interfaces, which can also be implemented by other concrete services hosted on other service providers. This enables more flexibility and ease in choosing and changing service providers, and finally allows us to move towards a model of discovering and using e-services dynamically.

There are various possible ways to describe business level conversations. The conversation definition language WSCL has chosen a formal approach, based on XML.

WSCL is defined by an XML schema. WSCL 1.0 is a simple conversation definition language[1]. Its purpose is to provide and define the minimal set of concepts necessary to specify conversations. It is expected that WSCL will be extended for more complex web services framework, e.g. to include multi-party conversations, quality of service attributes, transactions, or composition of conversations.

A key advantage of formal specification languages such as WSCL is that they pave the way for the creation of service frameworks that will enable service implementers to offload the responsibility for conversation-related tasks to infrastructure. For example, a service developer could create WSCL specifications documenting their service's abstract interface, and make this documentation available to potential users. A software developer can then use WSCL compliant web services servers and tools that would provide such functionality as document type validation, conversation tracking, and message dispatching to the application logic. Ideally, software developers creating and using web services will be supported by tools that allow them to easily and quickly map the interactions outlined in the conversation definition to any existing applications and back-end logic, while separating cleanly between the conversational and the application logic. Without any formal definition of the conversations, such tool support will not be possible.

Conversation definitions may either be specified directly by service providers, or they may be specified by industry specific standards groups, as e.g. RosettaNet PIPs are specified by RosettaNet and used by participating enterprises[2]. A service provider can either provide conversation definitions directly to the users of the service, e.g. as part of a trading partner agreement, or register[3] them as UDDI tModels in a UDDI registry [5], where they may also be referred to by businessService entries, and are available to anybody who has access to the registry.

Figure 2 depicts a UML diagram of a simple purchase conversation definition from the perspective of the seller. A service that supports this conversation definition expects a conversation to begin with the receipt of a LoginRQ or a RegRQ document. Once the service has received one of these documents, it answers with a ValidLoginRS, a InvalidLoginRS, or a RegistrationRS, depending on the type and content of the message received. The WSCL specification for this conversation is given in Appendix B. Although this conversation is defined from the perspective of the seller, it can be used to determine the appropriate message types and sequences for both the seller and the buyer. The buyer simply derives his conversation definition by inverting the direction of the messages.

[1] WSCL 1.0 is derived from CDL as defined in the HP Service Specification Framework [1].
[2] In contrast to the specification guideline for RosettaNet PIPs, WSCL provides a formal language for specifying conversations, and its basic set of concepts is targeted at less complex frameworks.
[3] For details on how to register WSCL specifications see [7].

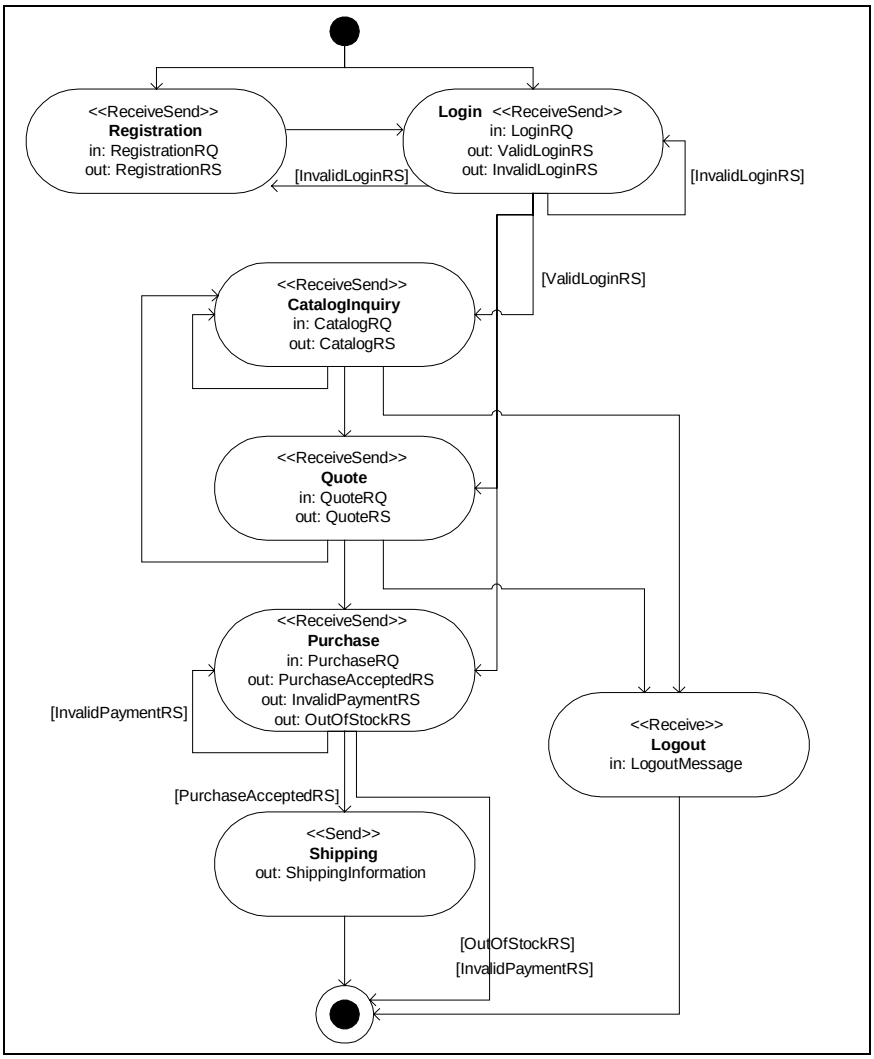

Figure 2: UML activity diagram of a conversation definition

Elements of a WSCL Specification

The complete schema for WSCL is listed in Appendix A. There are four main elements to a WSCL specification, as depicted in the UML model in Figure 3:

- *Document type descriptions* specify the types (schemas) of XML documents that the service can accept and transmit in the course of a conversation. The schemas of the documents exchanged are not specified as part of the WSCL specification document; the actual document schemas are separate XML documents and are referenced by their URL in the interaction elements of the conversation specification.
- *Interactions* model the actions of the conversation as document exchanges between conversation participants. WSCL currently supports five types of interactions: *Send* (the service sends out an outbound document), *Receive* (the service receives an inbound document), *SendReceive* (the service sends out an outbound document, then expects to receive an inbound document in reply), and *ReceiveSend* (the service

receives an inbound document and then sends out an outbound document). The *Empty* interaction does not contain any documents exchanged, it is just used for modeling the start and end of a conversation.

- *Transitions* specify the ordering relationships between interactions. A transition specifies a source interaction, a destination interaction, and optionally a document type of the source interaction as additional condition for the transition.
- The *Conversation* element lists all the interactions and transitions that make up the conversation. It also contains additional information about the conversation like its name, and with which interaction the conversation may start and end. A conversation can also be thought of as being one of the interfaces or public processes supported by a service. Yet in contrast to interfaces as defined by CORBA IDE or Java interfaces, conversations also specify the possible ordering of operations, i.e. the possible sequences in which documents may be exchanged.

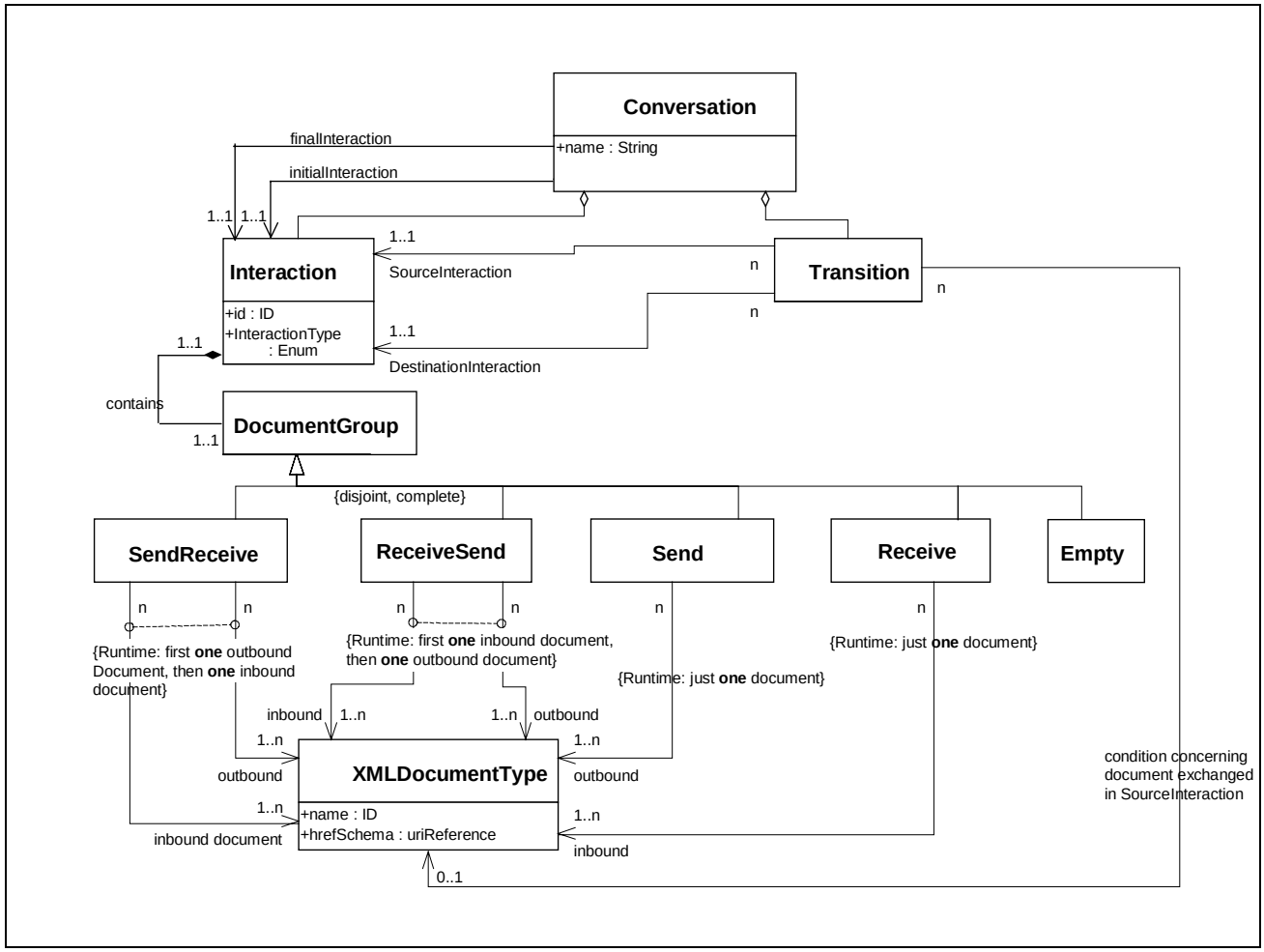

Figure 3: UML class diagram of WSCL

A conversation definition defines the conversation from the perspective of one of the participants. In most cases a conversation published in a service directory is the one defined from the perspective of the listener, i.e. the first interaction to happen is a Receive or ReceiveSend interaction. An initiator can derive its conversation definition from the conversation definition of the listener simply by converting Receive and ReceiveSend interactions into Send and SendReceive interactions, and vice versa. Essentially, two participants can successfully interact if the conversation definitions that they use are duals of each other.

Document Types

The interaction between service-consumer and service-provider is achieved through XML document exchange. A conversation definition language must be able to define all the input and output document types. The document type definitions refer to the schema that the document corresponds[4] to. They also serve to declare an id for the document that can be used within the rest of the conversation definition. For example, the following definition defines an input document that conforms to a purchase order schema defined in PurchaseOrderRQ.xsd.

```
<InboundXMLDocument
    hrefSchema="http://foo.org/PurchaseOrderRQ.xsd"
    id="PurchaseOrderRQ">
</InboundXMLDocument>
```

In a WSCL conversation definition, the document types are declared within the interaction definitions, either as an InboundXMLDocument or OutboundXMLDocument, depending on whether the document is expected as input or is produced as output in the interaction.

WSCL only references the schemas for the business payload of any messages. Of course, the business documents will be complemented by XML message header information for the exchange over the Internet, yet these message header schemas are defined by the protocol binding, and should not appear in the business documents referenced by the document types.

Interactions

An interaction is an exchange of one or two documents between a service and its client. WSCL only models business level interactions, i.e. it only specifies which business level documents are exchanged and does not model how this exchange is carried out by lower level messaging protocols. The actual messaging may involve more than one message per business document exchange, e.g. an HTTP-POST as well as an HTTP-RESPONSE per

[4]WSCL only supports XML schema specifications of payload, as schemas seem to become the prevailing means of describing data exchanged on the internet. Existing DTD specifications can easily be translated into XML schemas. WSCL, being defined as a very simple and basic conversation definition language, does not directly support the specification of non-XML payload like binary attachments.

business document. The messaging logic may even introduce additional transport level acknowledge messages, e.g. to achieve reliable messaging.

Currently the following interaction types are defined in WSCL: Receive and Send (one-way interactions), ReceiveSend and SendReceive (two-way interactions), Empty (see section on initial and final interactions).

One-way Interactions:

These interactions represent a single one-way message being sent or received by a participant. There are two sub-types of one-way interactions: Receive and Send. The Send interaction represents a document sent out by a participant, and on the other hand a Receive interaction represents a document being received by a participant.

The following example represents a Receive interaction that receives a login document:

```
<Interaction interactionType="Receive" id="LoginRegInput">
  <InboundXMLDocument id="LoginRequestData"
        hrefSchema=http://foo.org/LoginRequestData.xsd />
</Interaction>
```

A Receive interaction must contain one InboundXMLDocument element. A Send interaction must contain one OutboundXMLDocument.

Two-way Interactions:

Two-way interactions can have one of two forms, SendReceive or ReceiveSend, depending on whether the participant sends out a message for which it gets a response (SendReceive) or whether the participant responds to a request that it receives (ReceiveSend).

ReceiveSend Interactions: Each such interaction is the logical unit of receiving a request and then returning a response. The interaction is not complete until the response has been sent.

```
<Interaction interactionType="ReceiveSend" id="Quotation">
  <InboundXMLDocument id="PurchaseOrderRQ"
        hrefSchema=”http://foo.org/PurchaseOrderRQ.xsd”>
  </InboundXMLDocument>
  <OutboundXMLDocument" id="InvoiceRS"
        hrefSchema="http://foo.org/InvoiceRS.xsd>
  </OutboundXMLDocument>
</Interaction>
```

SendReceive Interaction: Each such interaction is the logical unit of sending a request and then receiving a response. The interaction is not complete until the response has been exchanged.


```
<Interaction interactionType="SendReceive" id="Payment">
  <OutboundXMLDocument id="Invoice"
            hrefSchema="http://foo.org/InvoiceRS.xsd">
  </OutboundXMLDocument>
  <InboundXMLDocument" id="Payment"
            hrefSchema="http://foo.org/Payment.xsd>
  </InboundXMLDocument>
</Interaction>
```

A *ReceiveSend* interaction can specify more than one OutboundXMLDocument. This allows the modeling of cases where there are multiple types of response messages that a service might return in response to a specific request. Having additional possible responses is mainly used for error messages. If a ReceiveSend interaction specifies more than one outbound document type, at run-time only one of them is being exchanged. For modeling the cases where a request message initiates several response messages, several interactions have to be used.

The following example shows a Purchase interaction that specifies two additional outbound document types for the cases of invalid payment and out of stock.

```
<Interaction interactionType="ReceiveSend" id="Purchase" />
  <InboundXMLDocument hrefSchema="http://conv123.org/PurchaseOrderRQ.xsd"
          id="PurchaseOrderRQ" />
  <OutboundXMLDocument  id="PurchaseOrderAcceptedRS"
          hrefSchema="http://conv123.org/PurchaseOrderAcceptedRS.xsd" />
  <OutboundXMLDocument  id="InvalidPaymentRS"
          hrefSchema="http://conv123.org/InvalidPaymentRS.xsd" />
  <OutboundXMLDocument  id="OutOfStockRS"
          hrefSchema="http://conv123.org/OutOfStockRS.xsd" />
</Interaction>
```

Likewise, a *SendReceive* interaction can specify more than one InboundXMLDocument.

On the abstraction level of WSCL, ReceiveSend and SendReceive interactions are simply shortcuts for modeling several Send and Receive interactions. If we consider possible bindings of interactions to transport protocols, having ReceiveSend and SendReceive interactions makes it easier to specify bindings to synchronous protocols (see section about combining WSCL and WSDL descriptions), though the definition of such bindings is outside the scope of WSCL .[5]

Transitions

A conversation can proceed from one interaction to another one as allowed by the permissible sequencing defined in the transition elements.

[5] Additional reasons for having ReceiveSend and SendReceive interactions come into play when considering extended conversation definition languages where QoS requirements are added to the interaction specifications.


```
<Transition>
     <SourceInteraction href="Invoice"/>
     <DestinationInteraction href="Receipt"/>
     <SourceInteractionCondition href=”InvoiceRS”/>
</Transition>
```

The *SourceInteraction* references an interaction that can precede the DestinationInteraction when the conversation is executed. Similarly, the *DestinationInteraction* references one of the interactions that can follow the SourceInteraction when the conversation is executed. All transitions together thus specify all possible sequences of the interactions.

The *SourceInteractionCondition* is an additional constraint on the transition, needed if the SourceInteraction specifies more than one possible document to be exchanged, and the type of document exchanged has an influence on the possible next interactions. The SourceInteractionCondition references an OutboundXMLDocument of the SourceInteraction in case it is a ReceiveSend interaction, or an InboundXMLDocument from the SourceInteraction in case it is a SendReceive interaction. If no SourceInteractionCondition is listed, then the DestinationInteraction can be triggered independent of the types of documents exchanged in the SourceInteraction. There is one important limitation to specifying transitions without SourceInteractionCondition: if there exists a transition from SourceInteraction A to DestinationInteraction B that specifies a SourceInteractionCondition, then it is not possible to also specify a transition from SourceInteraction A to DestinationInteraction B without a SourceInteractionCondition.[6]

Transitions define all the permissible orders of interactions, but they do not specify under which condition which permissible sequence is chosen. This is determined by the internal application logic of both participants, often based on back-end information. E.g., in the example in Appendix B, it is the buyer who decides if the CatalogInquiry interaction is followed by a Quote interaction or a Purchase interaction, by sending either a PurchaseRQ or QuoteRQ document. A PurchaseRQ document triggers the Purchase interaction, a QuoteRQ document triggers the Quote interaction.

The transitions defined in a conversation can be represented by a directed graph, or a UML activity diagram. A UML activity diagram represents interactions by action states. SourceInteractionConditions appear as conditions on the transitions. E.g. in Figure 2, after the Login interaction the client can either initiate another Login interaction, a Registration interaction, a CatalogInquiry interaction, a Quote interaction, or a purchase interaction, by sending either a RegistrationRQ, LoginRQ, CatalogRQ, QuoteRQ or PurchaseRQ document. Yet the last interactions are only allowed if the Login interaction ended with a ValidLoginRS document. The Registration and Login interactions are only allowed in case of an InvalidLoginRS.

[6] In WSCL 1.0 SourceInteractionConditions only allow to specify exactly one document type that must have been exchanged. Allowing more complex conditions like listing several document types or excluding document types would allow to specify less transitions.

Initial and Final Interactions

Part of defining the possible ordering of interactions is also the specification of the first and last interactions of a conversation. In WSCL this is done by the attributes initialInteraction and finalInteraction of the Conversation element. InitialInteraction references the id of the first interaction to be executed in the conversation. FinalInteraction references the id of the last interaction to be executed.

```
<Conversation name=”ExampleConversation”
   initialInteraction = “Login”
   finalInteraction = “Purchase” >
```

Of course, there might be more than one interaction with which the conversation may start or end. In the example of Figure 2 and Appendix B, the conversation may end after a CatalogInquiry interaction, after a Quote interaction, after a Purchase interaction in case the purchase cannot go through, or after the Shipping interaction. Also, the conversation may start with either a Registration or a Login interaction, depending on whether the client is already a registered user or not. In order to specify several possible start and end interactions, interactions of type *Empty* are used. In the example of Figure 2 an empty interaction Start is added, plus transitions from Start to Registration and Login. The attribute initialInteraction references this empty interaction Start. Also, an empty interaction End is introduced with various transitions to it, and is referenced in the attribute finalInteraction.

An interaction of interactionType *Empt*y is an interaction where no documents are exchanged. Its specification does not contain the subelements InboundXMLDocument and OutboundXMLDocument. Currently, the only place where empty interactions can be used in a conversation definition is for handling the situation of having several possible final or initial interactions. For an example of empty interactions see the conversation example in Appendix B.

Conversation

Each conversation definition has the root-element *Conversation*.

```
<?xml version="1.0" encoding="UTF-8"?>
<Conversation name="StoreFrontServiceConversation" version=”1.01”
      initialInteraction="Start"  finalInteraction="End"
      targetNamespace=”http://example.com/conversations/StoreFront101”
      hrefSchema=”http://example.com/schema_files/StoreFront101.wscl”
      description=”Conversation for a Store Front Service” >
  <ConversationInteractions>
      list of all the interactions
  </ConversationInteractions>
  <ConversationTransitions>
      list of all the transitions
  </ConversationTransitions>
</Conversation>
```


The Conversation element contains the following two sub-elements:

- *ConversationInteractions*: consists of a list of all Interaction elements.
- *ConversationTransitions*: consists of a list of all Transition elements.

The Conversation element contains the following attributes:

- *initialInteraction* and *finalInteraction*: reference the initial and final interactions of the conversation.
- *name*: name of a conversation. The name is the shared piece of information needed by both parties so they realize the same conversation type in their service implementation. This name would also appear in the header elements of the actual messages exchanged. WSCL does not specify how to name conversations. Conversation names may but need not be URIs. If a conversation is published in a UDDI directory, the conversation name could also be a tModel key.
- *version* (optional): version of the conversation. If no version number is given, then the name of the conversation must be unique.
- *targetNamespace* (optional): namespace of this conversation as it should be used when elements of this conversation are referenced in other XML documents.
- *hrefSchema* (optional): URL of the file containing this conversation definition.
- *description* (optional): textual description of the conversation.

Well-formed Conversation Definitions

In order for a conversation definition to be well-formed, it has to adhere to the conversation schema in Appendix A, and it has to fulfill the following restrictions:

- All interactions have to be reachable from the initial interaction, and the final interaction has to be reachable from every other interaction.
- If there exists a transition from an interaction A to an interaction B that does not specify any SourceInteractionCondition, then there may not be another transition from interaction A to interaction B that specifies a SourceInteractionCondition.

Conversations fulfilling these conditions are considered as well-formed. Only well-formed conversations should be published and used.

In addition, we also recommend that the final interaction and the transitions to the final interaction are defined in such a way that it is clear for each participant when a conversation is finished. The example in Figure 2 contains a Receive interaction Logout. The only purpose of this interaction is to signal the listener that the initiator is no longer interested in continuing the conversation, and that the listener does not have to wait for a PurchaseRQ or QuoteRQ that may or may not be sent.

In order for conversations to be deterministic, transitions should be unambiguous at all times. This can be achieved in two different ways, depending on the service framework and interaction protocol used:

- The protocol requires that the message headers either contain the id of the inbound or outbound document as specified in the WSCL description of the service, or uniquely identify the interaction as well as document concerned in some other ways: no

additional restrictions concerning the conversation specifications are necessary. Ids of inbound and outbound documents are unique within a conversation definition, therefore the transition and target interaction can be unambiguously identified at runtime.

- The messages exchanged do not carry any meta-information about the conversation, interactions and documents exchanged: the inbound or outbound documents that trigger the transition from the same source interaction with the same source interaction conditions to different destination interactions must have different schemas that differ in their first top elements. The transition and target interaction can be unambiguously identified based on the business payload of the message.

Mapping to Protocol Bindings and Error Handling

WSCL specifies the business payload being exchanged between two parties, and the order of these document exchanges. WSCL does not specify the message structure in which the payload is contained, nor the header fields of these messages. We can assume that the header fields indicate both parties involved in the message exchange, plus the name of the conversation and an id referencing the ongoing conversation instance. Yet all these details are defined by the communication protocol used.

When executing a conversation we can expect errors to happen on two levels: errors concerning the document content and errors concerning the specified conversations. Errors of the first type occur if a participant receives the correct type of document with the correct document structure, but invalid values (e.g. invalid user name and password). These types of errors need to be handled by the business logic, and the conversation definition and document schemas have to allow the communication of these errors (e.g. a document type for invalid login, or invalid login fields in a general login response document type).

Errors concerning the document types used, concerning the order of interactions, or concerning the conversations to be used, need to be handled by the communication protocol. The protocol might have predefined error messages that signal “unexpected document type” or “unknown conversation”. Also, the protocol has to define if a conversation instance continues or terminates after the exchange of an invalid document type, or what happens if one of the participants does not react for a long time.

WSCL does not specify how interactions are bound to a specific communication protocol, as protocol bindings are outside the scope of WSCL. There are two different approaches for providing binding information:

- The protocol binding is determined by a specific web services framework used, and is applicable to all services and all messages exchanged.
- A special protocol binding description language is used to describe the protocol bindings for the individual interactions and/or conversations. An example of such a protocol binding description language is the protocol binding part of WSDL [6].

A description of how to use WSDL to describe the protocol binding is given in the next section.

Using WSCL and WSDL to Describe Web Services

When describing web services we can distinguish between three main aspects (see also Figure 1):

- **Abstract interface** (public process, business model): the messages or documents (business payload) being exchanged and the order in which they are exchanged (choreography of the messages).
- **Protocol binding**: the protocols used for exchanging the documents.
- **Services**: the concrete service implementing a set of abstract interfaces and protocol bindings, and its location.

The following table shows how these three different aspects are covered by WSDL and WSCL. It is evident that the only overlap between WSDL and WSCL exists in the specification of the documents being exchanged:

		WSDL	**WSCL**
Abstract Interfaces	choreography	*out of scope*	Transition
	messages	Operation	Interaction
Protocol Bindings		Binding	*out of scope*
Concrete Services		Service	*out of scope*

Where WSDL and WSCL overlap, we can map the different terminology used as follows:

WSDL	**WSCL**
Port Type	Conversation
Operation: - One-way - Request-response - Solicit-response - Notification	Interaction[7]: - Receive - ReceiveSend - SendReceive - Send
Input	InboundXMLDocument
Output, Fault	OutboundXMLDocument
Names of Operation, Input, Output, Fault	Id of Interaction, InboundXMLDocument, OutboundXMLDocument
Message	URL of XML schema (WSCL delegates the specification of the payload entirely to an external XML schema, whereas WSDL directly uses XML data types)

There are three approaches for combining WSDL and WSCL descriptions of a web service:

[7] The interaction of type "Empty" does not appear in this list as it is only used for modeling the start and end state of conversations, and does not contain any documents exchanged.

1. *Adding protocol bindings in WSDL to a conversation described in WSCL*: If the abstract interface is already completely described by a WSCL document, we can go ahead and use the WSDL Binding elements to describe the protocol binding. Example:

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="MyStoreFrontService"
       targetNamespace=…
       xmlns:conv=”http://example.com/Conversations/StoreFront.cdl” >
  <binding name="StoreFrontServiceConversationBinding"
       type="conv:StoreFrontServiceConversation">
    <soap:binding style=”document”/>
    <operation name=”conv:Login”>
      <soap:operation soapAction=”Login”>
    </operation>
    <operation name=”conv:Registration”>
      <soap:operation soapAction=”Registration”>
    </operation>
    … for all other interactions
  </binding>

  <service name=”MyStoreFrontService”>
    <port name=”MyStoreFrontServiceAccessPoint”
       binding=”StoreFrontServiceConversationBinding”>
      <soap:Address location=”http://mystore.com/storefront” />
    </port>
  </service>
</definitions>
```

   In this example the attribute type in the element binding refers to the name of the conversation in the WSCL document describing this conversation. The names of the operations refer to the ids of the interactions, the type attribute of the binding refers to the name of the conversation in the WSCL document.

2. *Adding choreography to a WSDL port type description*: If a port type is already described by a WSDL document, we can describe the choreography in an additional WSCL document that only contains transition elements. The subelements SourceInteraction and DestinationInteraction refer to the names of operations from the WSDL document, and the subelement SourceInteractionCondition refers to an output or fault message of the operation.[8]

3. *Providing a full WSDL and WSCL description for the same port type / conversation*: We can also express which documents get exchanged by both, a WSDL port type and a WSCL conversation, with the conversation also describing the choreography used. The following example describes the interaction “Login” from the WSCL example in the appendix as a WSDL operation:

[8] WSCL uses attributes of type HREF to refer to other elements. Therefore, in order to refer to operations from a WSDL document the WSCL schema needs to be slightly adapted in order to accept also values of type QNAME.


```
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="MyStoreFrontService"
       targetNamespace=…
       xmlns:xsd1=”http://example.com/types/LoginRQ.xsd”
       xmlns:xsd2=”http://example.com/types/ValidLoginRS.xsd”
       xmlns:xsd3=”http://example.com/types/InvalidLoginRS.xsd” >

  <message name=”LoginRequestDocument”>
      <part name=”body” element=”xsd1:LoginRQElement” />
  </message>
  <message name=”ValidLoginDocument”>
      <part name=”body” element=”xsd2:ValidLoginRSElement” />
  </message>
  <message name=”InvalidLoginDocument”>
      <part name=”body” element=”xsd3:InvalidLoginRSElement” />
  </message>

  <portType name="StoreFrontServiceConversation" >
    <operation name=”Login”>
      <input name=”LoginRQ” message=”LoginRequestDocument”/>
      <output name=”ValidLoginRQ” message=”ValidLoginDocument”/>
      <fault name=”InvalidLoginRQ” message=”InvalidLoginDocument”/>
    </operation>
  </portType>
</definitions>
```

Extending WSCL

The WSCL specification contained in this proposal contains the smallest possible set of elements and attributes to describe conversations. This set is sufficient to model many of the conversations needed for web services. Yet there are more complex B2B interactions that need additional capabilities from a conversation definition language. Such additional requirements are e.g.:

- Defining document types that have non-XML content, e.g. binary attachments.
- Explicit description of roles of participants.
- Multiparty conversations with three or more participants or roles.
- Expressing timeouts and other quality of service characteristics of individual interactions.
- Expressing more complex SourceInteractionConditions, e.g. listing several documents, excluding documents, or even referencing the content of documents.
- Events, i.e. interactions that can occur at any time within a conversation instance.
- Recursive conversations, aggregating conversations into larger conversations.
- Subtyping and extending existing conversation definitions.

A proposal for some of these extensions can be found in [8].

References

[1] Hewlett-Packard, Co. *Service Framework Specification Version 2.0.* 2001, http://www.hp.com/go/e-speak

[2] *Simple Object Access Protocol (SOAP) 1.1*, W3C Note, May 2000

[3] *RosettaNet Implementation Framework RNIF, version 2.0*, RosettaNet Consortium, http://www.rosettanet.org

[4] *Message Service Specification, ebXML Transport, Routing & Packaging*, April

2001, EBXML, http://www.ebxml.org
[5] Ariba, International Business Machines Corporation, Microsoft Corporation, *UDDI Technical White Paper*, Sep 6, 2000.
[6] *Web Services Description Language (WSDL) 1.0.* URL: http://msdn.microsoft.com/xml/general/wsdl.asp
[7] Kuno, H., Lemon, M., Beringer, D. *Using WSCL in a UDDI Registry: UDDI Working Draft Best Practices Document.* May, 2001, http://www.hp.com/go/e-speak
[8] Beringer, D. *Extended CDL for modeling the choreography of RosettaNet PIPs.* Draft, March 2001. HP E-speak organization, to be published.

Appendix A: WSCL Schema Definition

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema  xsd:id="WSCL"  xmlns:xsd="http://www.w3.org/2000/10/XMLSchema"
    xsd:targetNamespace="http://www.e-speak.net/schema/WSCL"
    xsd:elementFormDefault="unqualified" xsd:attributeFormDefault="unqualified">

  <xsd:element name="Conversation">
   <xsd:complexType>
    <xsd:sequence>
        <xsd:element ref="ConversationInteractions"/>
        <xsd:element ref="ConversationTransitions"/>
    </xsd:sequence>
    <xsd:attribute name="name" type="xsd:string" use="required"/>
    <xsd:attribute name="version" type="xsd:string" use="optional"/>
    <xsd:attribute name="description" type="xsd:string" use="optional"/>
    <xsd:attribute name="targetNamespace" type="xsd:uriReference" use="optional"/>
    <xsd:attribute name="hrefSchema" type="xsd:uriReference" use="optional"/>
    <xsd:attribute name="initialInteraction" type="xsd:IDREF" use="required"/>
    <xsd:attribute name="finalInteraction" type="xsd:IDREF" use="required"/>
   </xsd:complexType>
  </xsd:element>

  <xsd:element name="ConversationInteractions">
   <xsd:complexType>
    <xsd:sequence>
        <xsd:element ref="Interaction" maxOccurs="unbounded"/>
    </xsd:sequence>
   </xsd:complexType>
  </xsd:element>

  <xsd:element name="Interaction">
   <xsd:complexType>
    <xsd:choice>
        <xsd:group ref="ReceiveSendDocumentGroup"/>
        <xsd:group ref="SendReceiveDocumentGroup"/>
        <xsd:group ref="ReceiveDocumentGroup"/>
        <xsd:group ref="SendDocumentGroup"/>
        <xsd:group ref="EmptyDocumentGroup"/>
    </xsd:choice>
    <xsd:attribute name="id" type="xsd:ID" use="required"/>
    <xsd:attribute name="interactionType" use="required">
        <xsd:simpleType>
            <xsd:restriction base="xsd:string">
                <xsd:enumeration value="SendReceive"/>
                <xsd:enumeration value="ReceiveSend"/>
                <xsd:enumeration value="Receive"/>
                <xsd:enumeration value="Send"/>
                <xsd:enumeration value="Empty"/>
            </xsd:restriction>
        </xsd:simpleType>
    </xsd:attribute>
   </xsd:complexType>
  </xsd:element>

  <xsd:group name="EmptyDocumentGroup"/>

  <xsd:group name="ReceiveSendDocumentGroup">
   <xsd:sequence>
    <xsd:element ref="InboundXMLDocument" />
    <xsd:element ref="OutboundXMLDocument" maxOccurs="unbounded"/>
   </xsd:sequence>
  </xsd:group>

  <xsd:group name="SendReceiveDocumentGroup">
```



```
    <xsd:sequence>
     <xsd:element ref="OutboundXMLDocument"/>
     <xsd:element ref="InboundXMLDocument" maxOccurs="unbounded"/>
    </xsd:sequence>
   </xsd:group>

   <xsd:group name="ReceiveDocumentGroup">
    <xsd:sequence>
     <xsd:element ref="InboundXMLDocument"/>
    </xsd:sequence>
   </xsd:group>

   <xsd:group name="SendDocumentGroup">
    <xsd:sequence>
     <xsd:element ref="OutboundXMLDocument"/>
    </xsd:sequence>
   </xsd:group>

   <xsd:element name="InboundXMLDocument" type="XMLDocumentType"/>

   <xsd:element name="OutboundXMLDocument" type="XMLDocumentType"/>

   <xsd:complexType name="XMLDocumentType">
    <xsd:attribute name="id" type="xsd:ID" use="required"/>
    <xsd:attribute name="hrefSchema" type="xsd:uriReference" use="optional"/>
   </xsd:complexType>

   <xsd:element name="ConversationTransitions">
    <xsd:complexType>
     <xsd:sequence>
          <xsd:element ref="Transition" minOccurs="0" maxOccurs="unbounded"/>
     </xsd:sequence>
    </xsd:complexType>
   </xsd:element>

   <xsd:element name="Transition">
    <xsd:complexType>
     <xsd:sequence>
          <xsd:element ref="SourceInteraction" />
          <xsd:element ref="DestinationInteraction" />
          <xsd:element ref="SourceInteractionCondition"  minOccurs="0" />
     </xsd:sequence>
    </xsd:complexType>
   </xsd:element>

   <xsd:element name="SourceInteraction">
    <xsd:complexType>
     <xsd:attribute name="href" type="xsd:IDREF" use="required"/>
    </xsd:complexType>
   </xsd:element>

   <xsd:element name="DestinationInteraction">
    <xsd:complexType>
     <xsd:attribute name="href" type="xsd:IDREF" use="required"/>
    </xsd:complexType>
   </xsd:element>

   <xsd:element name="SourceInteractionCondition">
    <xsd:complexType>
     <xsd:attribute name="href" type="xsd:IDREF" use="required"/>
    </xsd:complexType>
   </xsd:element>
  </xsd:schema>
```


Appendix B: Example WSCL Specification

The following XML document is a WSCL specification for the example conversation shown in Figure 2.

```
<?xml version="1.0" encoding="UTF-8"?>
<Conversation name="StoreFrontServiceConversation"
        initialInteraction="Start"  finalInteraction="End"  >
  <ConversationInteractions>
    <Interaction interactionType="ReceiveSend" id="Login">
        <InboundXMLDocument hrefSchema="http://conv123.org/LoginRQ.xsd"
               id="LoginRQ"/>
        <OutboundXMLDocument hrefSchema="http://conv123.org/ValidLoginRS.xsd"
               id="ValidLoginRS"/>
        <OutboundXMLDocument hrefSchema="http://conv123.org/InvalidLoginRS.xsd"
               id="InvalidLoginRS" />
    </Interaction>
    <Interaction interactionType="ReceiveSend" id="Registration">
        <InboundXMLDocument hrefSchema="http://conv123.org/RegistrationRQ.xsd"
               id="LoginRQ"/>
        <OutboundXMLDocument hrefSchema="http://conv123.org/RegistrationRS.xsd"
               id="RegistrationRS"/>
    </Interaction>
    <Interaction interactionType="Receive" id="Logout">
        <InboundXMLDocument hrefSchema="http://conv123.org/Logout.xsd"
               id="LogoutMessage"/>
        <OutboundXMLDocument hrefSchema="http://conv123.org/RegistrationRS.xsd"
               id="RegistrationRS"/>
    </Interaction>
    <Interaction interactionType="ReceiveSend" id="CatalogInquiry" >
        <InboundXMLDocument hrefSchema=http://conv123.org/CatalogRQ.xsd
               id="CatalogRQ"/>
        <OutboundXMLDocument hrefSchema="http://conv123.org/CatalogRS.xsd"
               id="CatalogRS" />
    </Interaction>
    <Interaction interactionType="ReceiveSend" id="Quote" >
        <InboundXMLDocument hrefSchema="http://conv123.org/QuoteRQ.xsd"
               id="QuoteRQ" />
        <OutboundXMLDocument hrefSchema="http://conv123.org/QuoteRS.xsd"
               id="QuoteRS" />
    </Interaction>
    <Interaction interactionType="ReceiveSend" id="Purchase" />
        <InboundXMLDocument hrefSchema="http://conv123.org/PurchaseOrderRQ.xsd"
               id="PurchaseOrderRQ" />
        <OutboundXMLDocument  id="PurchaseOrderAcceptedRS"
               hrefSchema="http://conv123.org/PurchaseOrderAcceptedRS.xsd" />
        <OutboundXMLDocument  id="InvalidPaymentRS"
               hrefSchema="http://conv123.org/InvalidPaymentRS.xsd" />
        <OutboundXMLDocument  id="OutOfStockRS"
               hrefSchema="http://conv123.org/OutOfStockRS.xsd" />
    </Interaction>
    <Interaction interactionType="Send" id="Shipping" >
            <OutboundXMLDocument   id="ShippingInformation"
               hrefSchema="http://conv123.org/ShippingInformation.xsd" />
    </Interaction>
    <Interaction interactionType="Empty" id="Start" />
    <Interaction interactionType="Empty" id="End" />
  </ConversationInteractions>

  <ConversationTransitions>
    <Transition>
      <SourceInteraction href="Start"/>
      <DestinationInteraction href="Login"/>
```



```
</Transition>
<Transition>
  <SourceInteraction href="Start"/>
  <DestinationInteraction href="Login"/>
</Transition>
<Transition>
  <SourceInteraction href="Registration"/>
  <DestinationInteraction href="Registration"/>
</Transition>
<Transition>
  <SourceInteraction href="Login"/>
  <DestinationInteraction href="Registration"/>
  <SourceInteractionCondition href="InvalidLoginRS"/>
</Transition>
<Transition>
  <SourceInteraction href="Login"/>
  <DestinationInteraction href="Login"/>
  <SourceInteractionCondition href="InvalidLoginRS"/>
</Transition>
<Transition>
  <SourceInteraction href="Login"/>
  <DestinationInteraction href="CatalogInquiry"/>
  <SourceInteractionCondition href="ValidLoginRS"/>
</Transition>
<Transition>
  <SourceInteraction href="Login"/>
  <DestinationInteraction href="Quote"/>
  <SourceInteractionCondition href="ValidLoginRS"/>
</Transition>
<Transition>
  <SourceInteraction href="Login"/>
  <DestinationInteraction href="Purchase"/>
  <SourceInteractionCondition href="ValidLoginRS"/>
</Transition>
<Transition>
  <SourceInteraction href="CatalogInquiry"/>
  <DestinationInteraction href="CatalogInquiry"/>
</Transition>
<Transition>
  <SourceInteraction href="CatalogInquiry"/>
  <DestinationInteraction href="Quote"/>
</Transition>
<Transition>
  <SourceInteraction href="Quote"/>
  <DestinationInteraction href="CatalogInquiry"/>
</Transition>
<Transition>
  <SourceInteraction href="Quote"/>
  <DestinationInteraction href="Purchase"/>
</Transition>
<Transition>
  <SourceInteraction href="Purchase"/>
  <DestinationInteraction href="Purchase"/>
  <SourceInteractionCondition href="InvalidPaymentRS"/>
</Transition>
<Transition>
  <SourceInteraction href="Purchase"/>
  <DestinationInteraction href="Shipping"/>
  <SourceInteractionCondition href="PurchaseAcceptedRS"/>
</Transition>
 <Transition>
  <SourceInteraction href="Shipping"/>
  <DestinationInteraction href="End"/>
```



```
    </Transition>
     <Transition>
      <SourceInteraction href="Purchase"/>
      <DestinationInteraction href="End"/>
      <SourceInteractionCondition href="InvalidPaymentRS"/>
    </Transition>
     <Transition>
      <SourceInteraction href="Purchase"/>
      <DestinationInteraction href="End"/>
      <SourceInteractionCondition href="OutOfStockRS"/>
    </Transition>
     <Transition>
      <SourceInteraction href="Quote"/>
      <DestinationInteraction href="Logout"/>
    </Transition>
     <Transition>
      <SourceInteraction href="CatalogInquiry"/>
      <DestinationInteraction href="Logout"/>
    </Transition>
     <Transition>
      <SourceInteraction href="Logout"/>
      <DestinationInteraction href="End"/>
    </Transition>
  </ConversationTransitions>
</Conversation>
```

