Using WSCL in a UDDI Registry 1.02
UDDI Working Draft Technical Note Document
May 5, 2001

This version:

http://www.uddi.org/pubs/wscl_TN_forUDDI_5_16_011.doc

Latest version:

http://www.uddi.org/pubs/wscl_TN_forUDDI_5_16_011.doc

Authors (alphabetically)**:**

Dorothea Beringer, Hewlett-Packard Company

Harumi Kuno, Hewlett-Packard Company

Mike Lemon, Hewlett-Packard Company

Contents

Management Overview

Electronic commerce is moving towards a vision of web-service based interactions, where corporate enterprises use web-services to interact with each other dynamically. For example, a service in one enterprise could spontaneously decide to engage a service fronted by another enterprise. In order for services to interact with each other dynamically, they must be able to do three fundamental things.

1. Clients must be able to discover services.
2. A service must be able to describe its abstract interfaces and protocol bindings so that clients can figure out how to invoke it.
3. A service must be able to describe the kinds of interactions (conversations) that it supports (e.g., that it expects clients to login before they can request a catalog) so that it can engage in complex exchanges with its clients.

The Universal Description Discovery and Integration (UDDI)[1-3] specifications address the first problem by defining a way to publish and discover information about Web services. The Web Services Description Language (WSDL)[4] addresses the second problem, defining a general purpose XML language for describing the interface and protocol bindings of network services. The Web Services Conversation Language (WSCL)[1][6] addresses the last problem, providing a standard way to model the public processes of a service, thus enabling network services to participate in rich interactions. Together, UDDI, WSDL, and WSCL enable developers to implement web services capable of spontaneously engaging in dynamic and complex inter-enterprise interactions.

In this document, we clarify the relationship between WSCL, UDDI, and WSDL. In particular, we describe how together these three components can be used to create an environment in which services can spontaneously discover each other and then engage in complicated interactions.

Current Situation

Web-services are much more loosely coupled than traditional distributed applications. This difference impacts both the requirements and usage models for web-services. Web-services are deployed on the behalf of diverse enterprises, and the programmers who implement them are unlikely to collaborate with each other during development. However, the purpose of web-services is to enable business-to-business interactions. Therefore, web-services must support very flexible, dynamic bindings. Web-services should be able to discover new services and interact with them dynamically without requiring programming changes to either service.

The prevalent model for web-service communication is that web-services will publish information about the specifications that they support. UDDI facilitates the publication and discovery of web-service information. The current version of WSDL (1.0) is an XML-based format that describes the interfaces and protocol bindings of web service functional endpoints. WSDL also defines the payload that is exchanged using a specific messaging protocol; SOAP is one such possible messaging protocol. However, neither UDDI nor WSDL currently addresses the problem of how a service can specify the sequences of legal message exchanges (interactions) that it supports.

The Web Services Conversation Language (WSCL) addresses this issue, providing an XML schema for defining legal sequences of documents that web-services can exchange. Web services interact by exchanging messages. Each message can be expressed as a structured document (e.g., using XML) that is an instance of some document type (e.g., expressed using XML Schema). A message may be wrapped (nested) in an encompassing document, which can serve as an

[1] WSCL is derived from the Conversation Definition Language (CDL).

envelope that adds contextual (delivery or conversation specific) information (e.g., using SOAP). We define a conversation to be a sequence of message exchanges between two or more services. We define a conversation specification to be a formal description of "legal" message type-based conversations that a service supports. The complete specification of WSCL 1.0 can be found at [6].

WSCL and WSDL are highly complimentary – WSDL specifies how to send messages to a service and WSCL specifies the order in which such messages can be sent. The advantage of keeping the two distinct is that doing so allows us to decouple conversational interfaces (represented by WSCL) from service-specific interfaces (represented by WSDL). This means that a single conversation specification can be implemented by any number of services, independent of the protocols supported by the various implementations.

Using WSCL Conversation Descriptions in UDDI Registries

Usage of WSCL

WSCL specifies the public interface to web-services, it does not specify how the conversation participants will handle and produce the documents received and sent. A conversation definition is thus service independent, and can be used by any number of services with completely different implementations. A conversation developer (e.g. a vertical standards body) can create a WSCL description of some conversation, and publish it in a UDDI directory. A service provider who wanted to create a service that supported that conversation description could create and document service endpoints that support the message sequencing specified by the WSCL document. Any software developer who wants to create an application using the published web-service can download the WSCL files describing the conversations supported, and implement the necessary interactions accordingly. Ideally, software developers creating and using web-services will be supported by tools that allow them to interpret conversations and 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 public and the private processes. Without any formal definition of the conversations, such tool support will not be possible.

Figure 1 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 RegistrationRQ 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 A. 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' halves of a conversation.

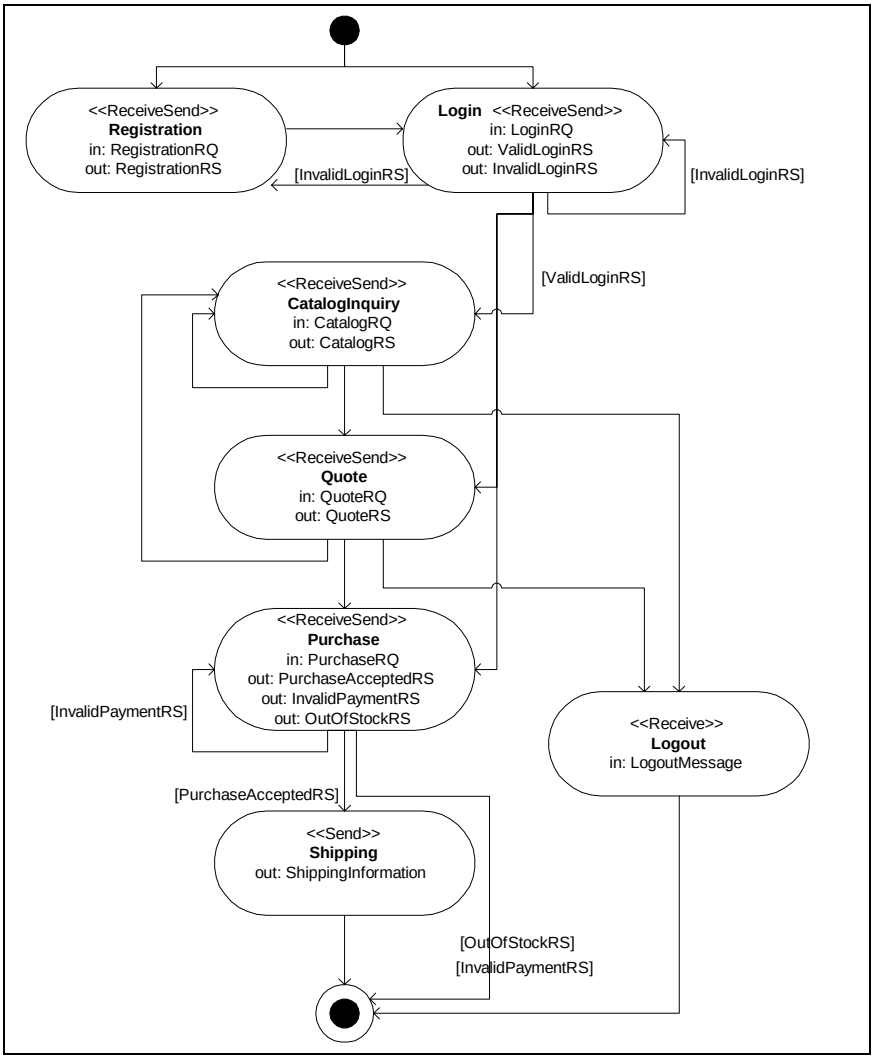

Figure 1: An example conversation depicted as a UML activity diagram. Interactions are represented by action states.

Authoring WSCL Conversation Specifications

There are four elements to a WSCL specification:

- *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. E.g., in figure 1, the interaction "Login" is defined as consisting of a the inbound message "LoginRQ" and either the outbound message "ValidLoginRS" or "InvalidLoginRS". For all these messages, the WSCL description also specifies the URL of the schema for the document type exchanged. WSCL currently supports four 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. E.g., in figure 1, after a "Purchase" interaction the conversation may either transition to the "Send", "Purchase" or final interaction, depending on the outbound document exchanged in the Purchase interaction.
- 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.

We now examine the XML representation of the conversation depicted in Figure 1, specified using WSCL (the full example can be found in Appendix A). The root *Conversation* element has attributes specifying a name for the conversation and the initial and final interactions, and contains two sub-elements: a *ConversationInteractions* element that contains a list of *Interaction* elements, and a *ConversationTransitions* element that consists of a list of *Transition* elements. WSCL does not restrict how to name conversations.

```
<?xml version="1.0" encoding="UTF-8"?>
<Conversation name="StoreFrontServiceConversation"
   initialInteraction="Start"  finalInteraction="End"  >
  <ConversationInteractions>
    <Interaction …  />
    <Interaction …  />
   …
  </ConversationInteractions>
  <ConversationTransitions>
    <Transition …  />
    <Transition …  />
   …
  </ConversationTransitions>
</Conversation>
```

The following XML code is a WSCL representation of the "Login" interaction, an interaction of type ReceiveSend that specifies several possible outbound documents, only one of which is exchanged at run-time.

```
<Interaction interactionType="ReceiveSend" id="Login">
  <InboundXMLDocument id="LoginRQ"
         hrefSchema="http://conv123.org/LoginRQ.xsd"  />
  <OutboundXMLDocument id="ValidLoginRS"
         hrefSchema="http://conv123.org/ValidLoginRS.xsd" />
  <OutboundXMLDocument id="InvalidLoginRS"
         hrefSchema="http://conv123.org/InvalidLoginRS.xsd" />
</Interaction>
```

Note that this interaction is assigned the identifier "Login." Also, note that each document type is assigned an identifier (e.g., "InvalidLoginRS"). The identifiers defined for the interactions and their documents are used when specifying the source and destination interactions for the conversation's transitions.

Transitions indicate the possible progression of the conversation between interactions. For example, the following XML code specifies that there is a possible transition from the "Login" to the "Quote" interaction, from the "Login" to the "CatalogInquiry" interaction, and from the "Login" to the "Registration" interaction. After sending out one of the outbound documents of the "Login" interaction, the service can receive e.g. a "RegistrationRQ", "CatalogRQ", or a "QuoteRQ"

document. In this specific case the next valid document also depends on the type of outbound document sent out in the “Login” interaction.

```
<Transition>
  <SourceInteraction href="Login"/>
  <DestinationInteraction href="Registration"/>
  <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>
```

Although WSCL specifies the valid inbound and outbound documents for an interaction, it does not specify how the conversation participants will handle and produce these documents. The WSCL specification of a conversation is thus service-independent, and can be used (and reused) by any number of service providers and service consumers.

The full conversation definition for Figure 1’s example conversation is listed in Appendix A.

Relevant UDDI Structures

A UDDI business registration is an XML document that describes a business entity and its web services. The UDDI XML schema defines four core types of service information: business information (such as business name and contact information), business service information (general technical and business descriptions of web services), binding information (specific information needed to invoke a service), and service specification information (associating a service’s binding information with the business service information it implements).

Programmers and programs can use the UDDI Business Registry to locate technical information about services, such as the protocols and specifications that they implement. More importantly, the UDDI Business Registry also serves as a registry for abstract (service-independent) specifications. Services can refer indirectly to the UDDI registrations for specifications they implement, which makes it straightforward to identify the business service information that represents a given service.

The UDDI *tModel* is a meta-data construct that uniquely identifies reusable service-related technical specifications for reference purposes. A service publishes tModelInstanceDetails, which is a list of tModelInfo elements that refer to the tModels that the service supports. A UDDI tModel data structure includes a unique key (*tModelKey* attribute), an *name* element, an optional description, and an overviewDoc element in which we can store a URL for the actual specification document.

We can use the *tModel* structure to register WSCL conversation specifications. Each tModel should be classified in the uddi-org:types taxonomy as being of type “wsclSpec” and must have an overviewDoc element whose overviewURL points to the actual WSCL document.

For example, suppose we wanted to register a WSCL specification of the “storefront” conversation depicted in Figure 1 in a UDDI registry. We would create a tModel entry within the UDDI registry that referred to the actual WSCL specification document in its *overviewDoc* element. The following XML code is a UDDI tModel reference for a WSCL specification for a service conversation.

```
<tModel authorizedName="XXXX" operator="YYYY" tModelKey="ZZZZ">
    <name>StoreFrontServiceConversation</name>
    <description xml:lang="en">
       WSCL description of a simple storefront conversation
    </description>
    <overviewDoc>
       <description xml:lang="eng">WSCL source document.</description>
       <overviewURL>http://example.com/schema_files/StoreFront101.wscl
       </overviewURL>
    </overviewDoc>
</tModel>
```

This storefront conversation tModel can now be referenced by the tModelInstanceInfo of any service that implements that conversation type:

```
<businessService>
  …
  <bindingTemplates>
    <bindingTemplate>
      …
      <accessPoint urlType="http">http://www.foo.com/</accessPoint>
        <tModelInstanceDetails>
          <tModelInstanceInfo tModelKey="ZZZZ">  …  </tModelInstanceInfo>
        </tModelInstanceDetails>
        …
    </bindingTemplate>
    …
  <bindingTemplates>
<businessService>
```

Relevant WSDL Structures

As noted before, WSCL specifications are conversation-specific. WSCL describes the structures (types) of documents a service expects to receive and produce, as well as the order in which document interchanges will take place, but does not specify how to dispatch received documents to the service. This is partially addressed by WSDL. WSDL documents describe part of the abstract interface as well as the protocol bindings of a network service. WSDL specifications that describe abstract protocol interfaces are reusable and thus can be registered as UDDI tModels [5].

When describing web services we can distinguish between three main aspects:

- **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[2]: 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:

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

[2] 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.

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.[3]

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:

```
<?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>
```

Referencing WSCL as well as WSDL Descriptions

The UDDI registry uses *businessService* data structures to store information about the specific services offered by a given business entity. Each *businessService* structure represents a logical service classification of a single *businessEntity* structure, and in turn uses *bindingTemplate* structures to hold the technical characteristics of the service. Each *bindingTemplate* structure includes *tModelInstanceDetails*, which is a list of *tModelInfo* structures. Each *tModelInfo* structure is a reference to a single *tModel.*

For example, suppose that we had a WSCL specification located at http://conversation-definitions/storefrontConversation.wscl, and WSDL specification containing binding information located at http://example.com/storefrontBinding.wsdl. We assume that in this example the WSDL document either references the interactions of the WSCL document in its binding elements, or that the two documents use the same names for interactions and document types, so the descriptions can be mapped as described above. UDDI tModels that refer to these documents might look like the following:

```
<tModel authorizedName="…" operator="…" tModelKey="1B345">
    <name>StoreFrontServiceConversation</name>
    <description xml:lang="en">
       WSCL description of a simple storefront conversation
    </description>
    <overviewDoc>
```

[3] 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.

```
       <description xml:lang="eng">WSCL source document.</description>
       <overviewURL>http://example.com/schema_files/StoreFront101.wscl
       </overviewURL>
    </overviewDoc>
</tModel>


<tModel authorizedName="…" operator="…"  tModelKey="2B345">
    <name>StoreFrontServiceBinding</name>
    <description xml:lang="en">
        WSDL description of a the binding of a storefront conversation
    </description>
    <overviewDoc>
     <description xml:lang="eng">WSDL source document.</description>
       <overviewURL>http://example.com/schema_files/StoreFront78.wsdl
    </overviewDoc>
</tModel>


<tModel authorizedName="…" operator="…"  tModelKey="3B345">
    <name>StoreFrontServiceInstance</name>
    <description xml:lang="en">
        WSDL description of a concrete service, listing bindings supported and
          addresses of these bindings
    </description>
    <overviewDoc>
     <description xml:lang="eng">WSDL source document.</description>
       <overviewURL>http://example.com/schema_files/StoreFrontMyService.wsdl
    </overviewDoc>
</tModel>
```

The UDDI *businessService* structure for a service that supports these tModels would look like the following:

```
<businessService>
  …
  <bindingTemplates>
    <bindingTemplate>
        …
      <accessPoint urlType="http">http://www.foo.com/</accessPoint>
      <tModelInstanceDetails>
         <tModelInstanceInfo tModelKey="1B345"> … </tModelInstanceInfo>
         <tModelInstanceInfo tModelKey="2B345"> … </tModelInstanceInfo>
         <tModelInstanceInfo tModelKey="3B345"> … </tModelInstanceInfo>
         …
      </tModelInstanceDetails>
     …
    <bindingTemplate>
  …
  <bindingTemplates>
<businessService>
```

Acknowledgements

We would like to thank Alan Karp, David Ehnebuske, Kannan Govindarajan, and Gregory Pogossiants for their comments and suggestions regarding this work. We owe special thanks to David Ehnebuske for his help with the WSDL-related issues.

References

[1] Ariba, International Business Machines Corporation, Microsoft Corporation, *UDDI Technical White Paper*, Sep 6, 2000.

[2] Boubez, T., Hondo, M., Kurt, C., Rodriguez, J., and Rogers, D., *UDDI Programmer's API 1.0*, Sep 20, 2000.

[3] Boubez, T., Hondo, M., Kurt, C., Rodriguez, J., and Rogers, D., *UDDI Data Structure Reference V1.0,* Sep 30, 2000.

[4] Web page: *Web Services Description Language (WSDL) 1.0*. URL: http://msdn.microsoft.com/xml/general/wsdl.asp

[5] Curbera, F., Ehnebuske, D., Rogers, D., *Using WSDL in a UDDI Registry 1.02*, http://www.uddi.org/bestpractices.html

[6] *Web-Services Conversation Language WSCL 1.0*, May 2001, http://www.hp.com/go/e-speak

Appendix A: Example WSCL Specification

The following XML document is an example of a WSCL specification for a conversation supported by a storefront service. Figure 1 shows the transition graph expressed in this conversation definition.

```
<?xml version="1.0" encoding="UTF-8"?>
<Conversation name="StoreFrontServiceConversation"
        targetNamespace=http://example.com/conversations/StoreFront101
        hrefSchema="http://example.com/schema_files/StoreFront101.wscl"
        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>
```