WS Choreography

Version 0-1, 13 June 2003

This version:
TBD
Latest version:
TBD
Previous Version:
Not Applicable
Authors (alphabetically):
David Burdett, Commerce One
Dan Gannon, Commerce One
Contributors (alphabetically):
Qiming Chen, Commerce One

Copyright

Abstract

This specification describes a formal method of defining a Choreography using a Choreography Definition Language. Choreographies describe the sequence and conditions in which messages are exchanged between independent processes, parties or organizations in order to realize some useful purpose, for example placing an order.

This differs from a Process Execution Language that can be used when there is a single organization or process in control that can issue commands to other processes to carry out all the actions or activities required.

If Choreographies are not defined and agreed between the organizations or processes involved, then those organizations and processes will not be able to successfully interoperate to realize their shared objectives.

(Note that this specification is a draft with some sections omitted.)

Status of this Document

This is the first version of the WS Choreography specification. Comments on this document are welcome …

This document may be updated, replaced or obsoleted by other documents at any time.

Table of Contents

1 Introduction

This specification describes a formal method of defining a Choreography using a Choreography Definition Language. Choreographies describe the sequence and conditions in which messages are exchanged between independent processes, parties or organizations in order to realize some useful purpose, for example placing an order.

Choreographies need to be defined when two or more organizations or processes need to cooperate as no single organization or process controls or manages the complete process. For example a Buyer cannot directly control what a Seller does and vice versa.

Note that this differs from a Process Execution Language that can be used when there is a single organization or process in control that can issue commands to other processes to carry out all the actions or activities required.

If Choreographies are not defined and agreed between the organizations or processes involved, then those organizations and processes will not be able to successfully interoperate to realize their shared objectives.

By providing a formal representation of a Choreography in an XML format, this specification allows the definition to be shared and therefore followed by all the organizations or processes that use it.

This specification is in two main parts:

- The first part describes how to define a Choreography in an abstract way that is independent of:
  - The format and packaging of the messages being exchanged, and
  - The technology used at each end to send and receive messages
- The second part describes how to bind the messages in a Choreography to WSDL and SOAP *(Ed: required but not included in this version spec)*.

Although bindings to WSDL and SOAP are provided, other bindings to other messaging technologies are possible although not described. This means that the abstract Choreography definition could be used to bind to messages in other formats such voice, paper or fax. These types of bindings are outside the scope of this specification.

1.1 Notational Conventions

The keywords "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

1.2 Namespaces

The namespaces used within this specification are as follows

Prefix	Namespace	Definition
tns	`com.commerceone.schemas/choreography/choreographydefinitions/choreographydefinitions.xsd`	Namespace for the WS Choreography schema definitions
xsd	`http://www.w3.org/2001/XMLSchema`	XML Schema Namespace

1.3 What's missing

Ed: The following is a partial list of items missing from this draft of the specification:

- *Bindings to WSDL and SOAP*
- *Generalized error handling, e.g. Message (document) errors, Process timeouts, transmission errors*
- *Composing new choreographies out of existing choreographies*
- *Relationships to Reliable Messaging and other protocols such as BTP and WS Transaction.*

1.4 Contents of this Document

The remainder of the specification provides:

- An *Overview* of the problems that this specification solves and the features of the WS-Choreography specification that help solve them
- An overview of the *Choreography XML Structure*. This provides a high-level overview of each of the components of a Choreography
- A description of the *Processing Rules* that apply
- A description of the type and semantics of each element and attribute in the Choroegraphy Schema

Appendices contain:

- The *Choreography* XML Schema definition
- An example XML Choreography Definition

2 Overview

2.1 The Problem

Two or more processes that need to co-operate by exchanging messages, must exchange messages in the same sequence if interoperability is to occur.

For example if a buyer sends a seller an order, the seller needs to know how to respond. Should they: a) return an order response message indicating the extent to which they can meet the order, b) just ship the goods and send an invoice or c) do something different. Interoperability problems will occur if the buyer is expecting an order response but gets an invoice instead.

This Choreography specification solves this problem by defining in an exchangeable XML format, the sequence, conditions and dependencies of sending one or more messages between the two or more processes or organizations involved in order to support some useful purpose.

2.2 Features

This rest of this section describes how to define choreographies that have the following features:

- *Reusability*. The same choreography definition is usable by different organizations operating in different contexts (industry, locale, etc) with different software (e.g. application software) and different message formats.
- *State Driven*. The specification defines how processes or organizations that take part in choreographies maintain where they are in the choreography by recording their state
- *Cooperative Organizations*. Choreographies define the sequence of exchanging messages between two (or more) independent organizations or processes by describing how they should cooperate
- *Verifiable*. The organizations or processes involved in a Choreography can use the Choreography definition to verify that a Choreography is being followed correctly. *(Ed: not defined in this spec but needed)*
- *Multi-Party*. The specification allows Choreography Definitions with any number of organizations or processes involved
- *Modular.* The Choreography specification includes an "import" facility that allows components of a specification that are defined separately to be imported

Each of these features is discussed in more detail in the following sections.

2.2.1 Reusability of Choreography Definitions

In an actual implementation messages flow between real services operated by real organizations. However this specification describes choreography definitions in a more abstract way that allows the choreography to be reused. The abstract concepts used include:

- *Roles*. A Role describes the type of behavior taken by the processes or organizations involved. For example an organization could take the role of a Buyer or a Seller when goods are being purchased.
- *States*. A State identifies the condition of a Role at a point in time. For example a Buyer State could be *OrderSent* after sending an Order message to a Seller.
- *Interactions*. An Interaction is the act of communicating information from one Role to another for a reason. For example sending an Order message from a Buyer to a Seller to request a purchase of goods or services
- *Message Families*. Message Families identify the set of messages that serve the same or similar purpose. For example a RosettaNet Order, a UBL Order, an EDI Order, all contain a request to purchase goods or services
- *Processes*. A process occurs as a result of some event such as a change of State. For example if a Seller's State becomes *OrderReceived* as a result of receiving an Order Message from a Buyer, then the change in state would trigger the Seller to carry out a process to check the Order.

Using these terms provides for reuse in multiple different contexts as follows:

- *Roles* can be mapped to specific organizations and processes, such as a specific Web Service.
- *States* can be mapped to specific conditions that occur in an organization, process or perhaps to specific values of an element in a message.
- *Interactions* allow the same message to be sent for multiple different reasons in multiple different choreographies. For example an Order could be sent to a Seller to request a purchase. It could also be sent to an archiving service for long-term retention. In both instances, the format of the message could be the same.
- *Message Families* can be mapped to specific message formats, for example the "Order" message family could be mapped to either a UBL or a RosettaNet Order Message
- *Processes* can be mapped to specific Web Services implemented by an organization

Without this type of abstraction, it will be necessary to define different choreographies for each implementation even if the basic purpose and the sequence of exchanging messages is the same, for example placing an Order.

2.2.2 State Driven Choreography Definitions

This specification assumes that:

- Each Role (e.g. a process or organization) participating in a choreography keeps records of their progress through the choreography by maintaining State information.
- The States values and their associated semantics, that each Role can take are defined in the Choreography definition and therefore can be shared by the organizations and roles involved
- State Changes can drive activities, for example: sending a message, or carrying out a process
- Changes in State are caused by events, for example: sending or receiving a message, or, carrying out a process.

The following example provides an illustration.

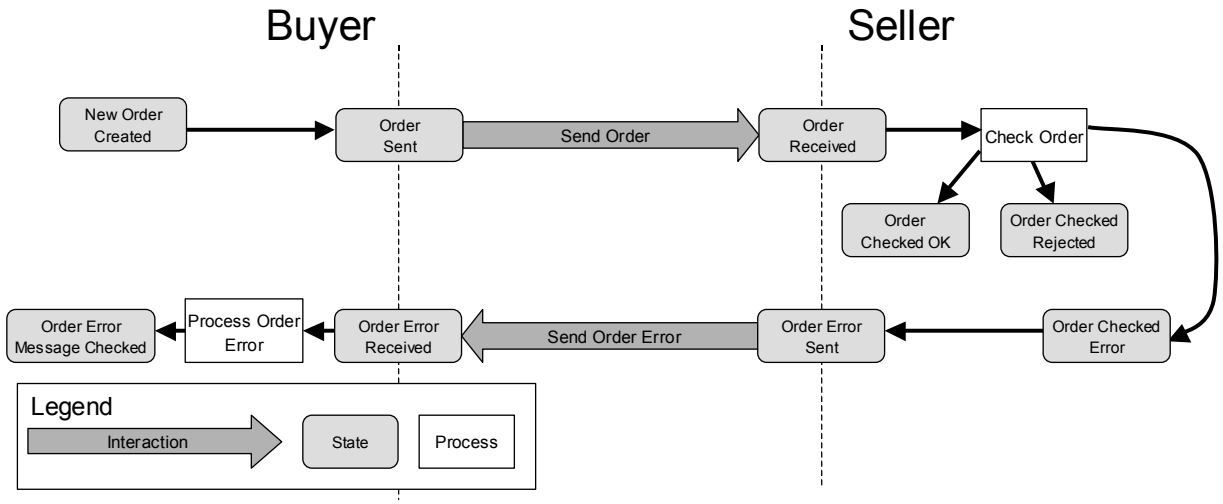


Figure 1: Illustration of the use of Roles, States, Processes and Interactions

This example illustrates the three main components of a choreography: *Interaction*s, *Processes* and *States* in an example where a Buyer *Role* places an order with a Seller *Role*. In this example there are just two *Interactions*:

- *SendOrder*. The Buyer sends an order to a seller, and
- *SendOrderError*. The Seller sends the Buyer an order error if there is a problem.

A more detailed explanation follows:

- The Buyer identifies a need to place an order. This results in a *NewOrderCreated* state occurring at the Buyer. How this state arrives is beyond the scope of this specification.
- The occurrence of a *NewOrderCreated* state results in the sending of a *SendOrder* message (called an *Interaction*) from the Buyer to the Seller.

- Once the *SendOrder* message is sent, the following state changes occur:
  - The Buyer state is changed to *OrderSent* once the *SendOrder* message is sent
  - The Seller state is changed to *OrderReceived* once the *SendOrder* message is received
- The occurrence of an *OrderReceived* state at the Seller causes the *CheckOrder* process to be executed by the Seller
- Once the *CheckOrder* process is complete, the Seller state is changed to one of the following:
  - *OrderCheckedOK* which means that no problems were found with the order and the Seller can satisfy the order,
  - *OrderCheckedRejected* which means that technically the Order was OK but it could not be satisfied by the Seller, or
  - *OrderCheckedError* which means there was some technical error with the Order that prevented the Order from being successfully processed.
- The occurrence of an *OrderCheckedError* state at the Seller causes the Seller to send a *SendOrderError* message to the Buyer
- Once the *SendOrderError* message is sent, the following state changes occur:
  - The Seller state is changed to *OrderErrorSent* once the *SendOrderError* message is sent
  - The Buyer state is changed to *OrderErrorReceived* once the *SendOrderError* message is received
- The occurrence of an *OrderErrorReceived* state at the Buyer causes the *ProcessOrderError* process to execute at the Buyer
- Once the *ProcessOrderError* process is complete, the Buyer state is changed to *OrderErrorMessageChecked*.

At this point the choreography is complete.

Note that the XML Choreography definition for this example is provided in Appendix B.1

2.2.3 Interactions, Reliable Messaging and Signals

Interactions describe the sending of a Message from one Role to another that result in a change of State. In practice, multiple additional messages may be exchanged between the roles as part of the binding of the choreography to a particular technology. Examples of these additional messages include:

- *Reliable Messaging* – These protocols involve the recipient of a message sending an acknowledgement to indicate the message was received as well as the sender re-sending the original message if no acknowledgement occurs.
- *Signals* – These are additional messages sent by the recipient of a message that indicate the processing of a message, for example that it has been validated or that processing has

started. They are general-purpose messages in that the same type of message can be sent as a response to many different Interactions.

Examples of these types of additional messages are shown in the diagram below.

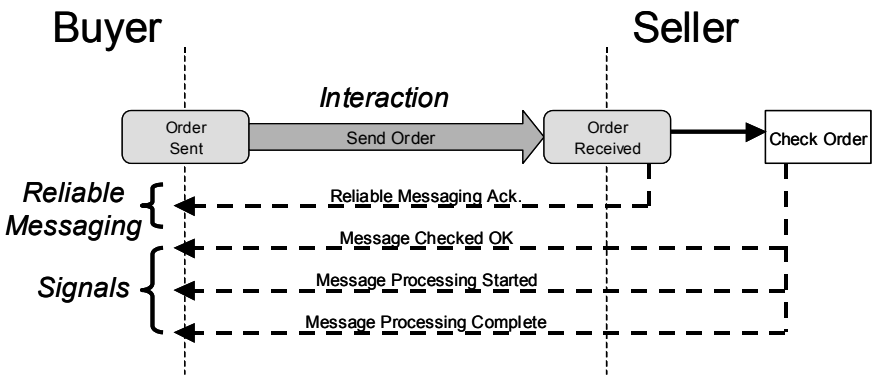


Figure 2: Interactions, Reliable Messaging and Signals

In this specification, only the first message – in this example the Send Order message – is defined as part of the Choreography. The other messages are part of the binding to the choreography to an implementation.

2.2.4 Cooperative Organizations

Internal Processes are executed by a single "Domain of Control" i.e. they are executed under single management control. Examples of single "Domains of Control" include:

- The processes running on a single hardware system or application
- A set of processes running on different hardware but controlled by a single set of rules defined in a Process Execution Language and executed by some Business Process Management software operated by an organization.

Choreographies differ from Internal Processes in that there are multiple Domains of Control. For example a Buyer would not normally allow a Seller to control how the Buyer's systems work and vice versa.

The consequence of this is that Choreographies specify how organizations must co-operate where no single organization is control.

2.2.5 Checking Choreography Progress

As no single organization or process is in control of a choreography it means each participant or process in a choreography must check that the choreography is progressing correctly by monitoring the messages that are being exchanged to ensure that they are being exchanged in the correct sequence.

This is achieved by:

- Carrying additional metadata in a message that identifies the interaction in a Choreography that is being sent
- Allowing one Role to inquire of another Role the state that they have reached.

If a Role discovers that a Choreography is not being followed correctly, then successful completion of the Choreography is not possible. In this case the Role that discovers a Choreography is not being followed informs the other Role(s) of the error.

(Ed: Note, none of this is defined in this spec but all are needed)

2.2.6 Multi-party Choreographies

Although many Choreographies involve just two organizations or processes, for example a Buyer and a Seller, this specification allows any number of different organizations or processes to take part.

2.2.7 Importing Definitions

The Choreography specification defines an Import facility that allows separately defined Roles, Message Families and Interactions to be imported and reused. This makes it easier to import choreography definitions defined elsewhere, for example by other organizations or standards bodies. *(Ed: We should probably include Process Definitions as something that can be imported as well)*

2.2.8 Extending Choreography Definitions

The Choreography specification allows one Choreography Definition to reference another Choreography Definition that it extends. The extension consists of adding additional Interactions and Processes to an existing Choreography definition. An example of an "extended" Choreography Definition is given in Appendix B.2 *(Ed: Not sure that this type of extensibility is the ideal way to go. Some type of Choreography composition would probably be a better alternative)*

2.2.9 Choreography Dependencies

The Choreography Specification describes how Choreography Definition can specify that it can only be used if some earlier Choreography Definition has been followed. For example, you could specify that an Order Status Inquiry Choreography can only be followed if an earlier Order Placement Choreography had been followed that the Order Status Inquiry could reference. An example of this type of Choreography dependency is given in Appendix B.3

2.3 Semantic Definitions

One of the features of this specification is to allow Choreography reuse by allowing the Choreography to be bound to solutions implemented by multiple different organizations using multiple different technologies.

For this to succeed an implementer needs clear definitions of what each part of the Choreography means otherwise the risk of incorrect implementations will significantly increase.

To solve this problem, this specification uses a *Description* element in many places so that the semantics of the Choreography are clear.

Full and proper use of the *Description* element is strongly recommended.

3 Choreography XML Structure

The following diagram illustrates the structure of a Choreography definition. It expands on the ideas of Roles, States, Interactions, Message Families and Processes described earlier.

The cardinality of each element or attribute is indicated as, for example, **`0..n`**. Cardinalities of **`1..1`** are the default. Note that this is not valid XML and is designed solely to provide an overview of the structure of a Choreography definition.

```
<Choreography defaultLanguage="Default language"
  <Description language="The language of the content of the description"0..1
               ref="URL of more detailed documentation"0..1 >0..n
      Description of the choreography in specified language
  </Description>
  <Description language="Alternate language for content of the description"0..1
               ref="URL of more detailed documentation"0..1 >
      Description of the choreography in an alternate language
  </Description>
  ...
  <!-- IMPORTS -->
  <!-- Imports, Roles, Message Families and Interactions - can occur in any order -->
  <Import namespace="URI of namespace of imported definitions"
          location="URL to be used as a hint to retrieve imports" 0..1/>0..n
  <Import ... />
  ...
  <!-- ROLES -->
  <Role name="Name of the role">0..n
    <Description>Semantics of the role</Description>0..n
    <State name="The name of a state the role can take">0..n
      <Description>Semantics of the state</Description>0..n
    </State>
    <State ... >
      ...
    </State>
    ...
  </Role>
  <Role ... >
    ...
  </Role>
  ...
```

```
  <!-- MESSAGE FAMILIES -->
  <MessageFamily name="Name of the Message Family"
                 urn="URN of the Message Family">
    <Description>Semantics of the Message Family</Description>0..n
  </MessageFamily>
  <MessageFamily ... >
    ...
  </MessageFamily>
  ...
  <!-- INTERACTION DEFINITIONS -->
  <InteractionDef name="Name of the Interaction Definition"
                  fromRole="Name of Role of sender of message"
                  toRole="Name of Role of receiver of message"
                  messageFamily="Name of the Message Family in the interaction">0..n
    <Description>Semantics of the Interaction Definition</Description>0..n
    <InteractionEndStates
                  fromState="State of the sending Role after message sent"
                  toState="State of the receiving Role after message received"/>
  </InteractionDef>0..n
  <InteractionDef ... >
    ...
  </InteractionDef>
  ...
  <!-- CHOREOGRAPHY DEFINITIONS -->
  <!-- One or more Choreography definitions are defined after definitions of roles, message
families and interactions -->
  <ChoreographyDefinition name="Name of the Choreography Definition"
                          urn="URN for the Choreography Definition">0..n
    <Description>Semantics/explanation of the Choreography Definition</Description>0..n
    <ExtendsChoreography urn="URN of the Choreography Definition being extended"/>0..1
    <DependsOnChoreography urn="URN of another Choreography Definition of which must have
occurred before this Choreography Definition can start"/>0..1
    <StartEndStates>
      <Start state="Name of a Start State"/>1..n
      <ConditionalEnd state="Name of a Conditional End State"/>0..n
      <CondtionalEnd .../>
      ...
      <End state="Name of an End State"/>0..n
      <End .../>
      ...
    </StartEndStates>
    <-- Interactions and Processes can occur in any order -->
    <Interaction name="Name of an Interaction Definition">0..n
      <Description>Semantics of the Interaction</Description>0..n
      <PreCondition Condition="Boolean combination of states which if present, cause Interaction
to occur."/>
    </Interaction>
    <Process name="Name of the process"
             role="Name of the role that executes the process">0..n
      <Description>Semantics of the process</Description>0..n
      <PreCondition condition="Boolean combination of states which if present, cause the Process
to occur."/>
      <ProcessEndState state="State of the Role after the process is complete"/>1..n
    </Process>
    <Process ... >
      ...
    </Process>
    <Interaction ... >
      ...
    </Interaction>
    ...
  </ChoreographyDefinition>
  <ChoreographyDefinition ...>
```

```
  <!-- Multiple ChoreographyDefinitions are allowed in one Choreography description -->
    ...
  </ChoreographyDefinition>
  ...
</Choreography>
```

4 Processing Rules

(Ed: To be completed. Sections to include:

- *Validation rules – over and above the schema validation*
- *How imports work*
- *How bindings work.*

Also need a section on WSDL binding.)

5 Schema Description

This section describes the elements within the Choreography Schema in alphabetical order.

5.1 Choreography

A single Choreography XML document contains definitions of common *Roles*, *Message Families* and *Interactions* that are used by one or more *Choreography Definitions*.

Recording more than one *Choreography Definition* in a *Choreography* file allows multiple *Choreography Definitions* to share the same *Roles*, *Message Families* and *Interactions*.

For example a simple order placement choreography could consist of sending an order from a Buyer to a Supplier with the Supplier just returning an error if the order could not be processed.

A more complex example could consist of the same messages but optionally followed by a ChangeOrder message that allows the Buyer to change the order after it was originally placed.

Both these variations of placing an order could accept the same *Message Families*, and use the same *Interactions* between the same *Roles*.

At a high level a *Choreography* contains:

- A required *defaultLanguage* attribute that specifies the default language used within *Description* elements within the *Choreography*
- Zero or more *Description* elements that provide an overall description of the complete *Choreography*
- Zero or more of the following elements in any order:

- An *Import* element that allows *Role*, *Message Family* or *Interaction* definitions to be included from a remote location
- A *Role* element that defines the *Roles* that take part in the Choreographies being defined and the *States* that the *Roles* are allowed to take
- A *MessageFamily* element that defines a Message Family that is used within the Choreography
- An *InteractionDef* element that defines an interaction between the *Roles*

A valid Choreography file must have at least two *Roles*, one *Message Family* and one *InteractionDef*.

5.1.1 Choreography@defaultLanguage

The *detaultLanguage* attribute is of type `xsd:language`. It defines the default language to be used by all the *Description* elements within the Choreography document unless over-ridden by the *language* attribute in a *Description* element.

5.1.2 Example

The following is an example of a *Choreography* element.

```
<Choreography defaultLanguage="us-en" xmlns="..."
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  ...
</Choreography>
```

5.2 ChoreographyDefinition

The *Choreography Definition* element defines a single Choreography. The *Choreography Definition* element contains:

- A required *name* attribute that is a unique identifier for the *Choreography Definition* within the *Choreography*. This can be used as an abbreviation when referencing the *Choreography Definition* from elsewhere in the *Choreography*
- A required *urn* attribute that contains a URI that uniquely identifies the *Choreography Definition*
- Zero or more *Description* elements that SHOULD contain definitions of the semantics of the *Choreography Definition*
- An optional *ExtendsChoreography* element that identifies another *Choreography Definition* that this *Choreography Definition* extends
- An optional *DependsOnChoreography* element that identifies another *Choreography Definition* that must have been followed before this *Choreography Definition* can be followed.

- A required *StartEndStates* element that defines the *States* that cause the start or indicate the end of the *Choreography Definition*
- One or more *Interaction* and *Process* elements in any order that indicate the sequence and conditions of sending and processing of the messages in the *Choreography Definition*

It is strongly RECOMMENDED that every *Choreography Definition* should include a clearly worded *Description* element to define the semantics.

5.2.1 ChoreographyDefinition@name

The *name* attribute is of type `xsd:ID`. It uniquely identifies the *Choreography Definition* within the *Choreography*.

5.2.2 ChoreographyDefinition@urn

The *urn* attribute is of type `xsd:uri`. It contains a URI that uniquely identifies the *Choreography Definition*.

5.2.3 Example

The following is an example of the *ChoreographyDefinition* element.

```
<ChoreographyDefinition name="OrderManagementChoreography1"
                        urn="http://example.com/choreographies/OrderManagement1">
  <Description>A simple Order Management Choreography that includes the sending of an order from
a Buyer to a Seller and the Seller returning an error if a problem is found.</Description>
  <StartEndStates>
    ...
  </StartEndStates>
</ChoreographyDefinition>
```

5.3 ConditionalEnd

The *ConditionalEnd* element is a child of the *StartEndStates* element.

The *ConditionalEnd* element identifies a *State* that may be the final *State* for a *Role* in a choreography.

For example, if a Buyer sends an Order to a Seller (see Figure 1:) then the Buyer's state becomes *OrderSent*. This may be the final state for the Buyer unless the seller discovered a problem with the Order Message. In this case, *OrderSent* is a "Conditional" end state for the Buyer as the state might but need not change. See also the *End* element.

5.3.1 ConditionalEnd@state

The *state* attribute is of type `xsd:IDref`. It contains the value of a *name* attribute on a *State* element of a role.

5.3.2 Example

The following is an example of the *ConditionalEnd* element.

```
<ConditionalEnd state="OrderSent"/>
```

5.4 DependsOnChoreography

The *DependsOnChoreography* element identifies another *Choreography Definition* that must have been followed before this *Choreography Definition* can be followed.

For example, you can specify that an Order Status Inquiry Choreography can only be followed if an earlier Order Placement Choreography had been followed that could be referenced. An example of this type of dependency is given in Appendix B.3

5.4.1 DependsOnChoreography@urn

The *urn* attribute is of type `xsd:uri`. It contains the value of a *urn* attribute on a *Choreography Definition*.

5.4.2 Example

The following is an example of the *DependsOnChoreography* element.

```
<DependsOnChoreography urn=" http://example.com/choreographies/OrderManagement1"/>
```

5.5 Description

The *Description* element is used to provide descriptions and semantics of the following elements in a Choreography: *Choreography*, *Roles*, *States*, *Message Families*, *Interaction Definitions*, *Choreography Definitions*, *Interactions* and *Processes*.

The *Description* element contains:

- An optional *language* attribute that specifies the language used in the element content
- An optional *ref* attribute that contains a URL to a more detailed human readable description or specification of the choreography component

- Element content that contains human readable description of the element

If no *language* attribute is present then the content of the *Description* element MUST contain a description in the language specified by the *defaultLanguage* attribute on the *Choreography* element.

Whenever the *Description* element is included in the definition of a component, it can occur zero or more times. The inclusion of at least one *Description* element is strongly recommended. If more than one *Description* element is present within a Choreography component, then each *Description* element should have a different value for the *language* attribute.

5.5.1 Description@language

The *language* attribute is of type `xsd:language`. It defines the language to be used by the element content of the *Description* element.

5.5.2 Description@ref

The *ref* attribute is of type `xsd:uri`. It contains a URL to a more detailed human readable description or specification of the Choreography component.

If present, the *ref* attribute should also reference a human readable document in the same language as the element content.

5.5.3 Example

The following is an example of the *Description* element.

```
<Description language="en-uk"
             ref="http://www.example.com/ChoreographySpecs/en-uk/OrderManagement.htm">
  This section contains a set of choreographies for Order Management
</Description>
```

5.6 End

The *End* element is a child element of the *StartEndStates* element.

The *End* element identifies a state that is a final state for a role in a choreography.

For example, if a Seller processes a *SendOrder* message received from a Buyer (see Figure 1:) and the SendOrder is valild, then the Seller's state becomes *OrderCheckedOK*. This is an "end" state as no further processes are dependent on it. See also the *ConditionalEnd* element.

5.6.1 End@state

The *state* attribute is of type `xsd:IDref`. It contains the value of a *name* attribute on a *State* element of a role.

5.6.2 Example

The following is an example of the *End* element.

```
<End state="OrderCheckedOK"/>
```

5.7 ExtendsChoreography

The *ExtendsChoreography* element identifies another *Choreography Definition* that this *Choreography Definition* extends. An extended choreography works by:

- Including the *Roles*, *States*, *Interactions* and *Processes* of the referenced *Choreography Definition*
- Specifying additional *Roles*, *States*, *Interactions* and *Processes* that extend the original *Choreography Definition*
- Specifying a new set of *StartEndStates* that applies to the combined *Choreography Definitions*.

An example of this type of dependency is given in Appendix B.2.

5.7.1 ExtendsChoreography@urn

The *urn* attribute is of type `xsd:uri`. It contains the value of a *urn* attribute on a *Choreography Definition*.

5.7.2 Example

The following is an example of the *ExtendsChoreography* element.

```
<ExtendsChoreography urn="http://example.com/choreographies/OrderManagement1"/>
```

5.8 Import

The *Import* element allows *Role*, *Message Family* or *Interaction Definitions* to be imported into a Choreography. It works in essentially the same way as the Import capability of WSDL. The *Import* element contains:

- A *namespace* attribute that identifies the namespace used for the imported definitions, and
- A *location* attribute that provides a hint for the physical location of the definitions.

(Ed: Issue, how do you handle conflicts when the name attribute on an imported definition is the same as the name attribute on another imported definition or directly included definition)

5.8.1 Import@namespace

The *namespace* attribute is of type `xsd:uri`. It contains the namespace of the imported definitions.

5.8.2 Import@location

The *location* attribute is of type `xsd:uri`. It contains a hint for the physical location of the definitions.

5.8.3 Example

The following is an example of the *Import* element.

```
<Import namespace="http://example.com/choreographies/OrderManagement/Roles"
        location="http://example.com/choreographies/OrderManagement/Roles"/>
```

5.9 Interaction

An *Interaction* element describes:

- The sending of a message in a *Message Family* from one *Role* to another, and
- The *PreConditions* that must exist before the *Interaction* can occur.

An Interaction always results in a change of state of the Roles that send and receive the message. Interactions do not include additional messages associated with the binding of an Interaction to an implementation. See section 2.2.3

It contains:

- A required *name* attribute that identifies the *InteractionDefinition* that describes the *Message Family* of the message being sent as well as the resulting states of the *FromRole* and the *ToRole* of the roles that send and receive the message
- Zero or more *Description* elements that provide a description of the semantics of the *Interaction*.
- A required *PreCondition* element that defines the conditions that must exist before the *Interaction* can occur.

5.9.1 Interaction@name

The *urn* attribute is of type `xsd:uri`. It contains the value of a *urn* attribute on a *Choreography Definition*.

5.9.2 Example

The following is an example of the *Interaction* element.

```
<Interaction name="SendOrder">
  <Description>Send the order to the seller</Description>
  <PreCondition condition="NewOrderCreated"/>
</Interaction>
```

5.10 InteractionDef

An *InreractionDef* element provides a reusable definition of an Interaction – i.e. the sending of a message from one role to another. It contains:

- The sending and receiving Roles, using the *fromRole* and *toRole* attributes
- The *Message Family* of the message being sent using the *messageFamily* attribute,
- Zero or more *Description* elements that provide a description of the semantics of the *Interaction Definition*
- The state of the sending and receiving roles using the *InteractionEndStates* element.

5.10.1 InteractionDef@name

The *name* attribute is of type `xsd:ID`. It uniquely identifies the *Interaction Definition* within the *Choreography*.

5.10.2 InteractionDef@fromRole

The *fromRole* attribute is of type `xsd:IDref`. It contains a reference to the *Role* that is sending the message.

5.10.3 InteractionDef@toRole

The *toRole* attribute is of type `xsd:IDref`. It contains a reference to the *Role* that is to receive the message.

5.10.4 InteractionDef@messageFamily

The *messageFamily* attribute is of type `xsd:IDref`. It contains a reference to the *Message Family* that is being sent in the *Interaction*.

5.10.5 Example

The following is an example of the *InteractionDef* element.

```
<InteractionDef name="SendOrder" fromRole="Buyer" toRole="Seller" messageFamily="Order">
  <Description>Send the order From the Buyer to the Seller</Description>
  <InteractionEndStates fromState="OrderSent" toState="OrderReceived"/>
</InteractionDef>
```

5.11 InteractionEndStates

The *InteractionEndStates* element defines the state of the *fromRole* and *toRole* that result from the *Interaction* occurring. It contains:

- A required *fromState* attribute that defines the state of the *fromRole*, and
- A required *toState* attribute that defines the state of the *toRole*.

(Ed: We might want to extend the idea of InteractionEndStates to include additional "error" states, such as: TransmissionErrorState, i.e. the Message could not be sent, DeliveryErrorState, i.e. the message could not be delivered with certainty as, for example, a reliable messaging acknowledgement was not received, and TimeoutErrorState, i.e. the expected response message was not received within some time.)

5.11.1 InteractionEndStates@fromState

The *fromState* attribute is of type `xsd:IDref`. It contains a reference to the *State* the *fromRole* takes after the message has been sent.

The *fromState* must be a state that belongs to the *fromRole*.

5.11.2 InteractionEndStates@toState

The *toState* attribute is of type `xsd:IDref`. It contains a reference to the *State* the *toRole* takes after the message has been received.

The *toState* must be a state that belongs to the *toRole*.

5.11.3 Example

The following is an example of the *InteractionEndStates* element.

```
<InteractionEndStates fromState="OrderSent" toState="OrderReceived"/>
```

5.12 MessageFamily

A *Message Family* identifies a set of messages that serve the same purpose. For example a RosettaNet Order, a UBL Order, an EDI Order, etc are all requests to purchase goods or services.

Interaction Definitions use *Message Families* as they allow the same *Choreography Definition* to be reused with different detailed message content.

The *Message Family* element contains:

- A required *name* attribute that is a unique identifier for the *Message Family* within the XML Choreography Document. This can be used as an abbreviation for the *Message Family* elsewhere
- A required *urn* attribute that contains a URI that uniquely identifies the *Message Family*
- Zero or more *Description* elements that SHOULD contain definitions of the semantics of the *Message Family.*

5.12.1 MessageFamily@name

The *name* attribute is of type `xsd:ID`. It uniquely identifies the *Message Family* within the Choreography XML document.

5.12.2 MessageFamily@urn

The *urn* attribute is of type `xsd:uri`. It contains a URN that uniquely identifies the *Message Family*.

5.12.3 Example

The following is an example of the *MessageFamily* element.

```
<MessageFamily name="Order" urn="http://example.com/MessageFamilies/OrderManagement/Order">
  <Description>Messages in this family contain information to convey a request to purchase goods
or services</Description>
</MessageFamily>
```

5.13 PreCondition

The *PreCondition* element describes the conditions that must exist before an *Interaction* or a *Process* can occur. It contains a Boolean expression consisting of a combination of *States* that must be true. For example `"OrderSent and OrderStatusCheckRequired"`.

(Ed: need to provide a precise grammar of what Boolean operations, parentheses, etc, are allowed)

5.13.1 PreCondition@condition

The *condition* attribute is of type `xsd:string`. It contains a Boolean expression consisting of a combination of *States*.

5.13.2 Example

The following is an example of the *PreCondition* element.

```
<PreCondition condition="OrderSent and OrderStatusCheckRequired"/>
```

5.14 Process

A *Process* element describes an activity or other process carried out by a *Role*. Processes occur as a result of a change of *State*. For example if a Seller's state becomes *OrderReceived* as a result of receiving an Order Message, then the Seller would carry out a process to check the Order.

The *Process* element contains:

- A required *name* attribute that identifies the Process within the *Choreography*
- A required *role* attribute that identifies the *Role* that carries out the process
- Zero or more *Description* elements that provide the semantics of the process
- A required *PreCondition* element that describes the conditions that must exist before the process can start, and
- One or more *ProcessEndState* elements that describe the possible states of the *Role* once the *Process* is complete.

5.14.1 Process@name

The *name* attribute is of type `xsd:ID`. It uniquely identifies the *Process* within the Choreography XML document.

5.14.2 Process@role

The *role* attribute is of tyle `xsd:IDref`. It identifies the Role that carries out the *Process*.

5.14.3 Example

The following is an example of the *Process* element.

```
<Process name="ProcessOrderStatusRequest" role="Seller">
  <Description>Process the Order Status Request and check if it is OK or in error</Description>
  <PreCondition condition="OrderStatusRequestReceived"/>
  <ProcessEndState state="OrderStatusRequestProcessedOK"/>
  <ProcessEndState state="OrderStatusRequestProcessedError"/>
  ...
</Process>
```

5.15 ProcessEndState.

The *ProcessEndState* element defines one of the possible states of a *Process* once it is complete. It consists of a single required *state* attribute.

(Ed: Do we want to extend ProcessEndState to include error states such as ProcessFailedState – the state of the Choreography if the process failed or crashed, and ProccessTimeoutError – the state of the Choreography if the process did not respond after some time.)

5.15.1 ProcessEndState@state

The *state* attribute is of type `xsd:IDref`. It references the *name* attribute on a *State* element. The *State* element referenced must be one of the states of the *Role* that executes the *Process*

5.15.2 Example

The following is an example of the *ProcessEndState* element.

```
<ProcessEndState state="OrderStatusRequestProcessedError"/>
```

5.16 Role

A *Role* identifies the type of activity taken by one of the organizations or processes that are participating in the *Choreography Definition*, for example a Buyer or Seller.

A *Role* consists of:

- A required *name* attribute that identifies a *Role* within the *Choreography*
- Zero or more *Description* elements that defines the semantics of the *Role*
- One or more *State* elements that list the possible states that the *Role* may take.

5.16.1 Role@name

The *name* attribute is of type `xsd:ID`. It uniquely identifies the *Role* within the *Choreography*.

5.16.2 Example

The following is an example of a *Role* element.

```
<Role name="Seller">
  <Description>This role represents the seller of goods or services</Description>
  <State name="OrderReceived"/>
  <State name="OrderCheckedOK"/>
  ...
</Role>
```

5.17 Start

The *Start* element identifies a *State* that, if it occurs, causes the Choreography to start. There must be at least one *Start State* in every *Choreography Definition*.

5.17.1 Start@state

The *state* attribute is of type `xsd:IDref`. It references a name *attribute* of a *State* element.

5.17.2 Example

The following is an example of a *Start* element.

```
<Start state="NewOrderCreated"/>
```

5.18 StartEndStates

The *StartEndStates* element identifies the states that indicate when the *Choreography Definition* must start as well as the states that indicate the *Choreography Definition* is complete.

It contains:

- Zero or more *Description* elements, to provide additional explanation about the Start and End States
- One or more *Start* elements that indicate the states that cause the *Choreography Definition* to start
- Zero or more *ConditionalEnd* elements that indicate a state that may be the final state of the *Choreography Definition*, and
- Zero or more *End* elements that indicate a state that, if reached, is a final state of the *Choreography Definition*.

Note that there must be at least one *ConditionalEnd* or *End* state for each *Role* that participates in the Choreography.

5.18.1 Example

The following is an example of the *StartEndStates* element.

```
<StartEndStates>
  <Start state="NewOrderCreated"/>
  <ConditionalEnd state="AcceptOrderSent"/>
  <ConditionalEnd state="RejectOrderSent"/>
  ...
  <End state="OrderErrorSent"/>
  <End state="OrderCheckedOK"/>
  ...
</StartEndStates>
```

5.19 State

A *State* element describes one of the states or conditions that a *Role* can take when participating in a *Choreography Definition*. For example a Buyer could have the state *OrderSent* after they send an order to a Seller.

It contains:

- A *name* attribute that uniquely identifies the *State* within the *Choreography*
- Zero or more *Description* elements that provide the semantics of the *State.*

5.19.1 State@name

The *name* attribute is of type `xsd:ID`. It uniquely identifies the *State* within the Choreography.

5.19.2 Example

The following is an example of the *State* element.

```
<State name="OrderSent"/>
```

6 References

[RFC2119] IETF RFC 2119. Key words for use in RFCs to Indicate Requirement Levels. S. Bradner, Harvard University, March 1997 http://www.ietf.org/rfc/rfc2119.txt

To be completed

Appendix A Choreography Schema (Normative)

The definitions for the Choreography Schema follow. The Schema Definition is in six parts:

- Choreography
- Description
- ImportType
- InteractionDefType
- MessageFamilyType
- RoleType

The last three "type" schema definitions are designed so that *InteractionDefTypes, MessageFamilyTypes* and *RoleTypes* can be independently defined and included using Import definitions.

The *DescriptionType* is separately defined so that a common *Description* element definition can be included in all the Schema definitions.

A.1 Choreography Schema

The following contains the Choreography Schema definition.

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- edited with XML Spy v4.4 U (http://www.xmlspy.com) by David Burdett (Commerce One) -->
<!--  David Burdett & Daniel Gannon (Commerce One) -->
<!-- Copyright Commerce One Operations Inc. (c) 2003. All rights reserved -->
```



```
<xsd:schema
targetNamespace="com.commerceone.schemas/choreography/choreographydefinitions/choreographydefini
tions.xsd"
xmlns="com.commerceone.schemas/choreography/choreographydefinitions/choreographydefinitions.xsd"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified">
 <xsd:annotation>
   <xsd:documentation xml:lang="en">
   Schema for Choreographies.
         </xsd:documentation>
 </xsd:annotation>
 <xsd:include schemaLocation="Description.xsd"/>
 <xsd:include schemaLocation="RoleType.xsd"/>
 <xsd:include schemaLocation="MessageFamilyType.xsd"/>
 <xsd:include schemaLocation="InteractionDefType.xsd"/>
 <xsd:complexType name="ImportType">
   <xsd:annotation>
     <xsd:documentation>Schema for importing external definitions of Roles and Message
Families</xsd:documentation>
   </xsd:annotation>
   <xsd:attribute name="namespace" type="xsd:anyURI" use="required"/>
   <xsd:attribute name="location" type="xsd:anyURI" use="optional"/>
 </xsd:complexType>
 <xsd:element name="Choreography">
   <xsd:annotation>
     <xsd:documentation>A Choreography can contain one or more Choreography Definitions
describing the interactions that can occur sent between roles.</xsd:documentation>
   </xsd:annotation>
   <xsd:complexType>
     <xsd:complexContent>
       <xsd:extension base="ChoreographyType">
         <xsd:attribute name="defaultLanguage" type="xsd:language" use="required">
           <xsd:annotation>
             <xsd:documentation>Default language for all Description elements unless overidden on
the individual element</xsd:documentation>
           </xsd:annotation>
         </xsd:attribute>
       </xsd:extension>
     </xsd:complexContent>
   </xsd:complexType>
 </xsd:element>
 <xsd:complexType name="ChoreographyType">
   <xsd:annotation>
     <xsd:documentation>Contains the high level definitions of Roles, Message Families and
Interactions. There must be at least two Roles and one Message Family in a Choreography file.
Role and Message Family definitions can be "imported" from files referenced by an Import
element.</xsd:documentation>
   </xsd:annotation>
   <xsd:sequence>
     <xsd:element name="Description" type="DescriptionType" minOccurs="0" maxOccurs="unbounded">
       <xsd:annotation>
         <xsd:documentation>Describes information regarding the whole definition of the complete
choreography file.</xsd:documentation>
       </xsd:annotation>
     </xsd:element>
     <xsd:choice maxOccurs="unbounded">
       <xsd:element name="Import" type="ImportType" minOccurs="0">
         <xsd:annotation>
           <xsd:documentation>Import specifies a file containing either Role, Message Family or
Interaction definitions.</xsd:documentation>
         </xsd:annotation>
       </xsd:element>
       <xsd:element name="Role" type="RoleType" minOccurs="0">
         <xsd:annotation>
```

```
        <xsd:documentation>Contains the Roles that can take part in the choreographies.
</xsd:documentation>
      </xsd:annotation>
    </xsd:element>
    <xsd:element name="MessageFamily" type="MessageFamilyType" minOccurs="0">
      <xsd:annotation>
        <xsd:documentation>Contains definitions of the message families, e.g. an Order Message
Family. A Message Family is a general name that can represent messages of different structures
that serve the same purpose. .</xsd:documentation>
      </xsd:annotation>
    </xsd:element>
    <xsd:element name="InteractionDef" type="InteractionDefType" minOccurs="0">
      <xsd:annotation>
        <xsd:documentation>An Interaction defines the sending of a message from one role to
another.</xsd:documentation>
      </xsd:annotation>
    </xsd:element>
  </xsd:choice>
  <xsd:element name="ChoreographyDefinition" maxOccurs="unbounded">
    <xsd:annotation>
      <xsd:documentation>Contains definitions of sequences of exchanges of interactions
(messages) between Roles and the processes that create or process them.</xsd:documentation>
    </xsd:annotation>
    <xsd:complexType>
      <xsd:complexContent>
        <xsd:extension base="ChoreographyDefinitionType"/>
      </xsd:complexContent>
    </xsd:complexType>
  </xsd:element>
 </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="ChoreographyDefinitionType">
 <xsd:annotation>
   <xsd:documentation>Defines: a) the start and end states of the choreography, b) the
interactions between the roles and c) the processes executed by the roles</xsd:documentation>
 </xsd:annotation>
 <xsd:sequence>
   <xsd:element name="Description" type="DescriptionType" minOccurs="0" maxOccurs="unbounded">
     <xsd:annotation>
       <xsd:documentation>A narrative that explains the purpose of the Choreography
Definition.</xsd:documentation>
     </xsd:annotation>
   </xsd:element>
   <xsd:element name="ExtendsChoreography" type="ExtendsChoreographyType" minOccurs="0">
     <xsd:annotation>
       <xsd:documentation>Identifies another choreography that this choreography definition
extends. All the interactions and processes in the identified choreography are automatically
included in this choreography.</xsd:documentation>
     </xsd:annotation>
   </xsd:element>
   <xsd:element name="DependsOnChoreography" type="DependsOnChoreographyType" minOccurs="0">
     <xsd:annotation>
       <xsd:documentation>Identifies another choreography an instance of which must have
occurred before this choreography can start.</xsd:documentation>
     </xsd:annotation>
   </xsd:element>
   <xsd:element name="StartEndStates" type="StartEndStatesType">
     <xsd:annotation>
       <xsd:documentation>Identifies the states that causes the Choreography to start and the
states which indicate the Choreography is finished</xsd:documentation>
     </xsd:annotation>
   </xsd:element>
   <xsd:choice maxOccurs="unbounded">
```

```
      <xsd:element name="Interaction">
        <xsd:annotation>
          <xsd:documentation>An interaction sends a message between two roles identifying the
MessageFamily exchanged. An interaction only occurs if certain preconditions
exist.</xsd:documentation>
        </xsd:annotation>
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="Description" type="DescriptionType" minOccurs="0"
maxOccurs="unbounded"/>
            <xsd:element name="PreCondition" type="PreConditionType">
              <xsd:annotation>
                <xsd:documentation>Contains a logical expression consisting of a combination of
role states that, if they evaluate to true, then the Interaction should
occur</xsd:documentation>
              </xsd:annotation>
            </xsd:element>
          </xsd:sequence>
          <xsd:attribute name="name" type="xsd:IDREF" use="required"/>
        </xsd:complexType>
      </xsd:element>
      <xsd:element name="Process" type="ProcessType">
        <xsd:annotation>
          <xsd:documentation>Describes a process executed by a role in terms of the preconditions
that must exist before the process can occur and the states that the role may have once the
process is complete</xsd:documentation>
        </xsd:annotation>
      </xsd:element>
    </xsd:choice>
  </xsd:sequence>
  <xsd:attribute name="name" type="xsd:ID" use="required">
    <xsd:annotation>
      <xsd:documentation>Is a descriptive name for the choreography. Choreography Names must be
unique within the Choreography File</xsd:documentation>
    </xsd:annotation>
  </xsd:attribute>
  <xsd:attribute name="urn" type="xsd:anyURI" use="required"/>
</xsd:complexType>
<xsd:complexType name="ExtendsChoreographyType">
  <xsd:annotation>
    <xsd:documentation>Identifies a choreography that this choreography
extends</xsd:documentation>
  </xsd:annotation>
  <xsd:sequence>
    <xsd:element name="Description" type="DescriptionType" minOccurs="0" maxOccurs="unbounded"/>
  </xsd:sequence>
  <xsd:attribute name="urn" type="xsd:anyURI" use="required"/>
</xsd:complexType>
<xsd:complexType name="DependsOnChoreographyType">
  <xsd:annotation>
    <xsd:documentation>Identifies a choreography that an instance of this choreography is
dependent on</xsd:documentation>
  </xsd:annotation>
  <xsd:sequence>
    <xsd:element name="Description" type="DescriptionType" minOccurs="0" maxOccurs="unbounded"/>
  </xsd:sequence>
  <xsd:attribute name="urn" type="xsd:anyURI" use="required"/>
</xsd:complexType>
<xsd:complexType name="StartEndStatesType">
  <xsd:annotation>
    <xsd:documentation> Defines the start and end states of a Choreography
Definition</xsd:documentation>
  </xsd:annotation>
```

```
    <xsd:element name="Description" type="DescriptionType" minOccurs="0" maxOccurs="unbounded"/>
    <xsd:element name           xOccurs="unbounded">
      <xsd:annotation>
        <xsd:documentation>Identifies a state which, if it exists, will trigger the start of the
choreography</xsd:documentation>
      </xsd:annotation>
      <xsd:complexType>
        <xsd:attribute name="state" type="xsd:IDREF" use="required"/>
      </xsd:complexType>
    </xsd:element>
    <xsd:element name="ConditionalEnd" minOccurs="0" maxOccurs="unbounded">
      <xsd:annotation>
        <xsd:documentation>Identifies a state which MAY be a final or end state of the
choreography. Additional interactions or processes can occur once this state is reached but need
not.</xsd:documentation>
      </xsd:annotation>
      <xsd:complexType>
        <xsd:attribute name="state" type="xsd:IDREF" use="required"/>
      </xsd:complexType>
    </xsd:element>
    <xsd:element name="End" minOccurs="0" maxOccurs="unbounded">
      <xsd:annotation>
        <xsd:documentation>Identifies a state which is a final or end state of the choreography.
No additional interactrions or process can validly occur once this state is
reached.</xsd:documentation>
      </xsd:annotation>
      <xsd:complexType>
        <xsd:attribute name="state" type="xsd:IDREF" use="required"/>
      </xsd:complexType>
    </xsd:element>
  </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="ProcessType">
  <xsd:annotation>
    <xsd:documentation>Defines a process</xsd:documentation>
  </xsd:annotation>
  <xsd:sequence>
    <xsd:element name="Description" type="DescriptionType" minOccurs="0" maxOccurs="unbounded">
      <xsd:annotation>
        <xsd:documentation>Describes the process executed by the role</xsd:documentation>
      </xsd:annotation>
    </xsd:element>
    <xsd:element name="PreCondition">
      <xsd:annotation>
        <xsd:documentation>Contains a logical expression consisting of a combination of role
states that, if they evaluate to true, then the Process should  be executed by the
role</xsd:documentation>
      </xsd:annotation>
      <xsd:complexType>
        <xsd:complexContent>
          <xsd:extension base="PreConditionType"/>
        </xsd:complexContent>
      </xsd:complexType>
    </xsd:element>
    <xsd:element name="ProcessEndState" maxOccurs="unbounded">
      <xsd:annotation>
        <xsd:documentation>References an internal state the role that executed the process can
have once the process is complete</xsd:documentation>
      </xsd:annotation>
      <xsd:complexType>
        <xsd:sequence>
          <xsd:element name="Description" minOccurs="0" maxOccurs="unbounded"/>
```

```
      </xsd:sequence>
      <xsd:attribute name="state" type="xsd:IDREF" use="required"/>
    </xsd:complexType>
  </xsd:element>
 </xsd:sequence>
 <xsd:attribute name="name" type="xsd:ID" use="required"/>
 <xsd:attribute name="role" type="xsd:IDREF" use="required"/>
</xsd:complexType>
<xsd:complexType name="PreConditionType">
 <xsd:annotation>
  <xsd:documentation>Describes the pre-conditions that apply to an Interaction or
Process</xsd:documentation>
 </xsd:annotation>
 <xsd:sequence>
  <xsd:element name="Description" type="DescriptionType" minOccurs="0" maxOccurs="unbounded"/>
 </xsd:sequence>
 <xsd:attribute name="condition" type="xsd:string" use="required"/>
</xsd:complexType>
</xsd:schema>
```

A.2 Description Schema

The following contains the Description Schema.

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- edited with XML Spy v4.4 U (http://www.xmlspy.com) by David Burdett (Commerce One) -->
<!-- Copyright Commerce One Operations Inc. (c) 2003. All rights reserved -->
<xsd:schema
targetNamespace="com.commerceone.schemas/choreography/choreographydefinitions/choreographydefini
tions.xsd"
xmlns="com.commerceone.schemas/choreography/choreographydefinitions/choreographydefinitions.xsd"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified"
attributeFormDefault="unqualified">
 <xsd:complexType name="DescriptionType" mixed="true">
  <xsd:annotation>
   <xsd:documentation>Describes a component of the Choreography</xsd:documentation>
  </xsd:annotation>
  <xsd:attribute name="language" type="xsd:language" use="optional">
   <xsd:annotation>
    <xsd:documentation>Overides the default language for the content of the
Description</xsd:documentation>
   </xsd:annotation>
  </xsd:attribute>
  <xsd:attribute name="ref" type="xsd:anyURI" use="optional">
   <xsd:annotation>
    <xsd:documentation>Contains the URL that can be resolved to discover further information
about what is being described</xsd:documentation>
   </xsd:annotation>
  </xsd:attribute>
 </xsd:complexType>
</xsd:schema>
```

A.3 ImportType Schema

The following contains the ImportType Schema.

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- edited with XML Spy v4.4 U (http://www.xmlspy.com) by Daniel Gannon (Commerce One) -->
<!-- Copyright Commerce One Operations Inc. (c) 2003. All rights reserved -->
<xsd:schema
targetNamespace="com.commerceone.schemas/choreography/choreographydefinitions/choreographydefini
tions.xsd" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns="com.commerceone.schemas/choreography/import/importtype.xsd"
elementFormDefault="qualified">
 <xsd:complexType name="ImportType">
   <xsd:annotation>
     <xsd:documentation>Schena for importing external definitions</xsd:documentation>
   </xsd:annotation>
   <xsd:attribute name="namespace" type="xsd:anyURI" use="required"/>
   <xsd:attribute name="location" type="xsd:anyURI" use="optional"/>
 </xsd:complexType>
</xsd:schema>
```

A.4 InteractionDefType Schema

The following contains the InteractionDefType Schema.

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- edited with XML Spy v4.4 U (http://www.xmlspy.com) by David Burdett (Commerce One) -->
<!-- Copyright Commerce One Operations Inc. (c) 2003. All rights reserved -->
<xsd:schema
targetNamespace="com.commerceone.schemas/choreography/choreographydefinitions/choreographydefini
tions.xsd" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns="com.commerceone.schemas/choreography/choreographydefinitions/choreographydefinitions.xsd"
elementFormDefault="qualified" attributeFormDefault="unqualified">
 <xsd:include schemaLocation="Description.xsd"/>
 <xsd:element name="InteractionDefList">
   <xsd:annotation>
     <xsd:documentation>A list of Interaction Definitions in a Choreography that may be Imported
into a Choreography Definition</xsd:documentation>
   </xsd:annotation>
   <xsd:complexType>
     <xsd:sequence>
       <xsd:element name="InteractionDef" type="InteractionDefType" maxOccurs="unbounded">
         <xsd:annotation>
           <xsd:documentation>Describes an Interaction in a Choreography. </xsd:documentation>
         </xsd:annotation>
       </xsd:element>
     </xsd:sequence>
   </xsd:complexType>
 </xsd:element>
 <xsd:complexType name="InteractionDefType">
   <xsd:annotation>
     <xsd:documentation>Defines an interaction between two roles.</xsd:documentation>
   </xsd:annotation>
   <xsd:sequence>
     <xsd:element name="Description" type="DescriptionType" minOccurs="0" maxOccurs="unbounded"/>
     <xsd:element name="InteractionEndStates">
       <xsd:annotation>
```

```
      <xsd:documentation>Defines the states that the from and to roles have once the
interaction is complete</xsd:documentation>
    </xsd:annotation>
    <xsd:complexType>
      <xsd:attribute name="fromState" type="xsd:IDREF" use="required"/>
      <xsd:attribute name="toState" type="xsd:IDREF" use="required"/>
    </xsd:complexType>
  </xsd:element>
 </xsd:sequence>
 <xsd:attribute name="name" type="xsd:ID" use="required">
  <xsd:annotation>
   <xsd:documentation>This is the identifier for the interaction. It must be unique within a
Choreography File</xsd:documentation>
  </xsd:annotation>
 </xsd:attribute>
 <xsd:attribute name="fromRole" type="xsd:IDREF" use="required">
  <xsd:annotation>
   <xsd:documentation>Indicates the role that is sending the message by referencing the Name
attribute for the role</xsd:documentation>
  </xsd:annotation>
 </xsd:attribute>
 <xsd:attribute name="toRole" type="xsd:IDREF" use="required">
  <xsd:annotation>
   <xsd:documentation>Indicates the role that is receiving the message by referenceing the
Name attribute for the Role</xsd:documentation>
  </xsd:annotation>
 </xsd:attribute>
 <xsd:attribute name="messageFamily" type="xsd:IDREF" use="required">
  <xsd:annotation>
   <xsd:documentation>Indicates the MessageFamily that is used in the interaction by
referencing the name attribute of the Message Family</xsd:documentation>
  </xsd:annotation>
 </xsd:attribute>
</xsd:complexType>
</xsd:schema>
```

A.5 MessageFamilyType Schema

The following contains the MessageFamilyType Schema.

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- edited with XML Spy v4.4 U (http://www.xmlspy.com) by David Burdett (Commerce One) -->
<!-- Copyright Commerce One Operations Inc. (c) 2003. All rights reserved -->
<xsd:schema
targetNamespace="com.commerceone.schemas/choreography/choreographydefinitions/choreographydefini
tions.xsd"
xmlns="com.commerceone.schemas/choreography/choreographydefinitions/choreographydefinitions.xsd"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified">
 <xsd:include schemaLocation="Description.xsd"/>
 <xsd:element name="MessageFamilyList">
  <xsd:annotation>
   <xsd:documentation>A list of Message Families in a Choreography that may be Imported into a
Choreography Definition</xsd:documentation>
  </xsd:annotation>
  <xsd:complexType>
   <xsd:sequence>
    <xsd:element name="MessageFamily" type="MessageFamilyType">
     <xsd:annotation>
```

```
        <xsd:documentation>A Message Family is used to group together messages that serve a
similar purpose. For example various different XML schema that each define an order would all be
part of the "Order" Message Family</xsd:documentation>
      </xsd:annotation>
    </xsd:element>
  </xsd:sequence>
 </xsd:complexType>
</xsd:element>
<xsd:complexType name="MessageFamilyType">
 <xsd:annotation>
  <xsd:documentation>The Message Family Type defines the structure of the Message
Family</xsd:documentation>
 </xsd:annotation>
 <xsd:sequence>
  <xsd:element name="Description" type="DescriptionType" minOccurs="0" maxOccurs="unbounded">
   <xsd:annotation>
    <xsd:documentation>This describes information regarding the MessageFamily, and what type
of information is included in the message</xsd:documentation>
   </xsd:annotation>
  </xsd:element>
 </xsd:sequence>
 <xsd:attribute name="name" type="xsd:ID" use="required">
  <xsd:annotation>
   <xsd:documentation>This is a descriptive name of the MessageFamily. It must be unique
within all Message Families within the Choreography.</xsd:documentation>
  </xsd:annotation>
 </xsd:attribute>
 <xsd:attribute name="urn" type="xsd:anyURI" use="required">
  <xsd:annotation>
   <xsd:documentation>This is the unique identifier for this MessageFamily</xsd:documentation>
  </xsd:annotation>
 </xsd:attribute>
</xsd:complexType>
</xsd:schema>
```

A.6 RoleType Schema

The following contains the RoleType Schema.

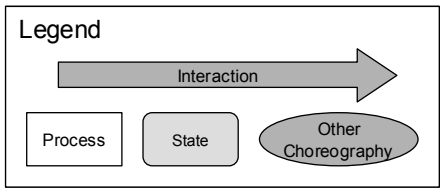
```
<?xml version="1.0" encoding="UTF-8"?>
<!-- edited with XML Spy v4.4 U (http://www.xmlspy.com) by David Burdett (Commerce One) -->
<!-- Copyright Commerce One Operations Inc. (c) 2003. All rights reserved -->
<xsd:schema
targetNamespace="com.commerceone.schemas/choreography/choreographydefinitions/choreographydefini
tions.xsd"
xmlns="com.commerceone.schemas/choreography/choreographydefinitions/choreographydefinitions.xsd"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified"
attributeFormDefault="unqualified">
 <xsd:include schemaLocation="Description.xsd"/>
 <xsd:element name="RoleList">
  <xsd:annotation>
   <xsd:documentation>A list of Roles in a Choreography that may be Imported into a
Choreography Definition</xsd:documentation>
  </xsd:annotation>
  <xsd:complexType>
   <xsd:sequence>
    <xsd:element name="Role" type="RoleType" maxOccurs="unbounded">
     <xsd:annotation>
```

```
        <xsd:documentation>Describes a Role in a Choreography Definition and the states it may
      </xsd:annotation>
    </xsd:element>
  </xsd:sequence>
 </xsd:complexType>
</xsd:element>
<xsd:complexType name="RoleType">
 <xsd:annotation>
  <xsd:documentation>Contains the structure of the Role</xsd:documentation>
 </xsd:annotation>
 <xsd:sequence>
  <xsd:element name="Description" type="DescriptionType" minOccurs="0">
   <xsd:annotation>
    <xsd:documentation>Describes the role, e.g. a buyer role is an organization or individual
that purchases goods or services</xsd:documentation>
   </xsd:annotation>
  </xsd:element>
  <xsd:element name="State" type="StateType" maxOccurs="unbounded">
   <xsd:annotation>
    <xsd:documentation>A state identifies the progress that has been reached in a
choreography. States arise as a result of sending, receiving or processing a
message.</xsd:documentation>
   </xsd:annotation>
  </xsd:element>
 </xsd:sequence>
 <xsd:attribute name="name" type="xsd:ID" use="required">
  <xsd:annotation>
   <xsd:documentation>Name is a descriptive name for the role, e.g Buyer. RoleNames MUST be
unique within a Choreography</xsd:documentation>
  </xsd:annotation>
 </xsd:attribute>
</xsd:complexType>
<xsd:complexType name="StateType">
 <xsd:sequence minOccurs="0">
  <xsd:element name="Description" type="DescriptionType"/>
 </xsd:sequence>
 <xsd:attribute name="name" type="xsd:ID" use="required">
  <xsd:annotation>
   <xsd:documentation>The names of a state must be unique within a Choreography
File.</xsd:documentation>
  </xsd:annotation>
 </xsd:attribute>
</xsd:complexType>
</xsd:schema>
```

Appendix B Example Choreography Definition (non-normative)

This appendix provides an example of a *Choreography*. It consists of four related *Choreography Definitions* in one XML document.

The following diagrams illustrate each of these choreographies using the following conventions.



Note that these conventions are purely illustrative. Other alternative graphical representations of the choreographies may be used.

B.1 Order Management 1

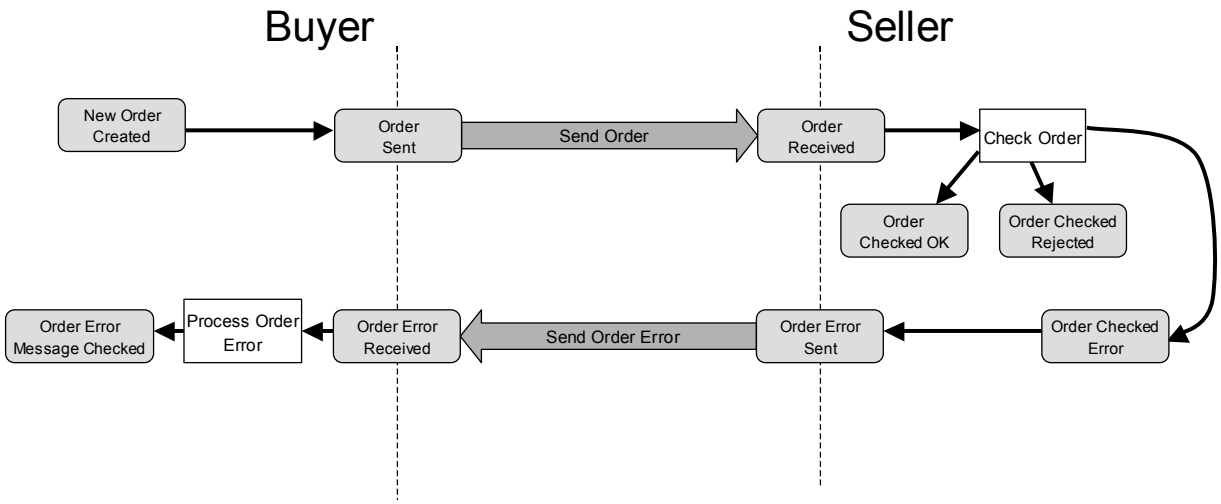


Figure 3: Order Create and Error Response

Order Management 1 consists of the sending of a single Order with an Error Message sent in reply only if a problem is detected.

B.2 Order Management 2

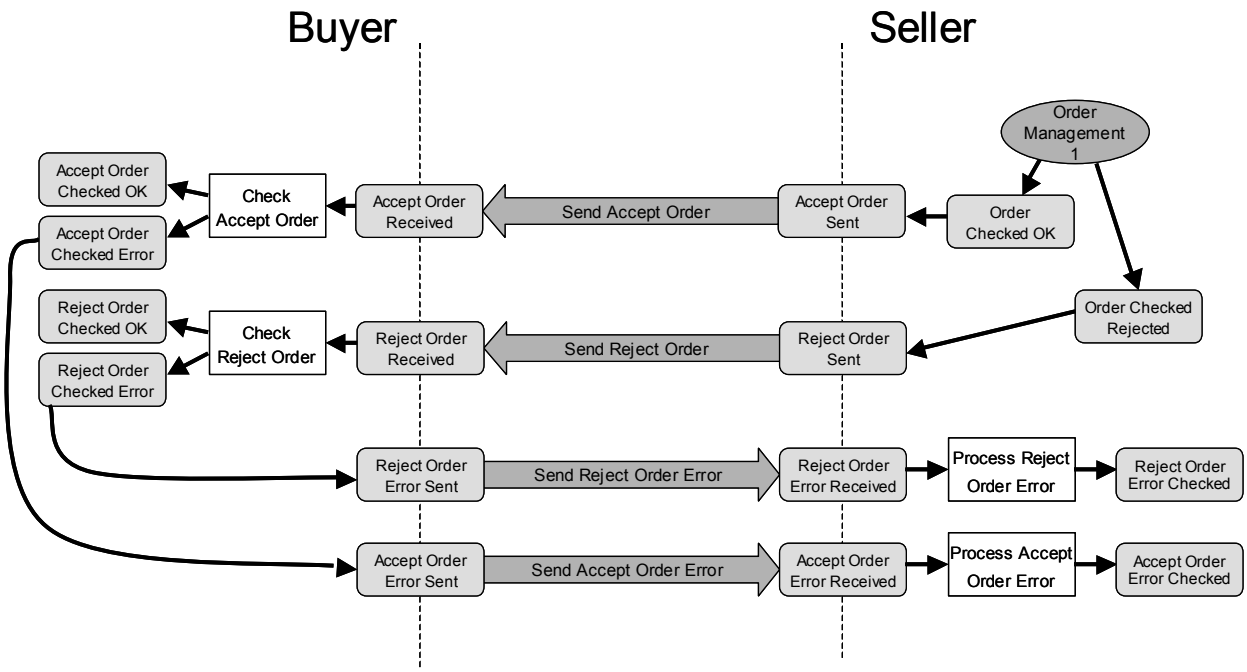


Figure 4: Order Create and Single Order Response – Extends Order Management 1

Order Management 2 "extends" Order Management 1 by adding the sending of either an “Accept Order” or a “Reject Order” message and associated error messages. As it is an “extension” it means that all the messages in Order Management 1 are included by reference.

B.3 Check Order Status

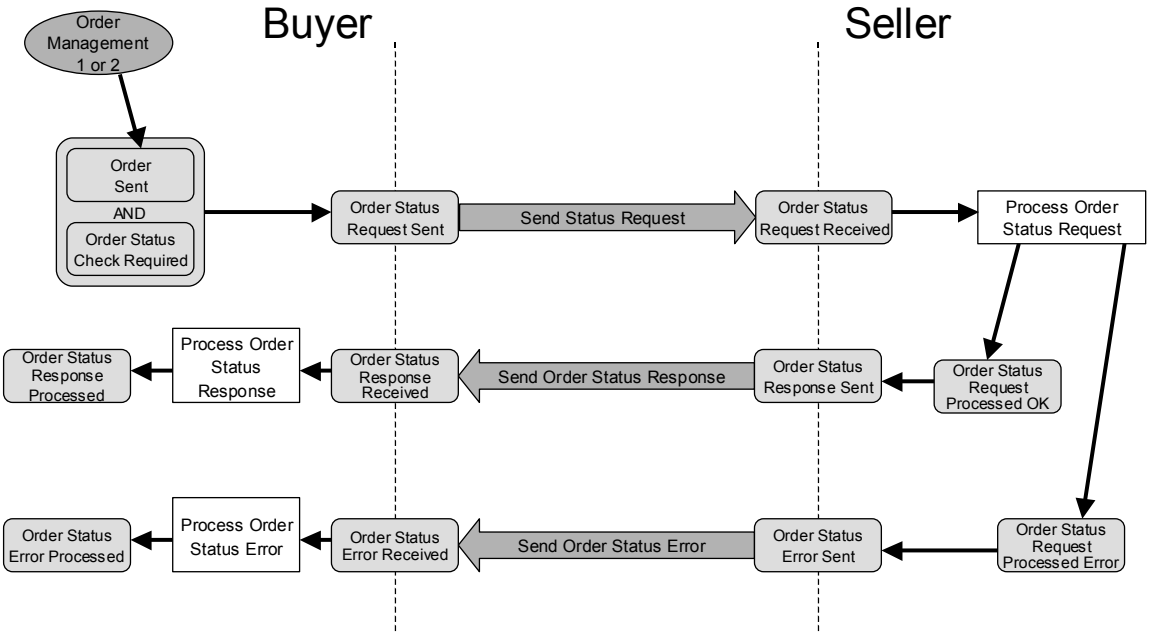


Figure 5: Check Order Status – Depends on Order Management 1 or Order Management 2

Check Order Status is a separate Choreography that is “dependent” on Order Management 1 or Order Management 2 choreography being followed earlier. It is used to determine the status of an order. As it is “dependent” on another choreography, it means that it can only be followed if an instance of the other choreography has already occurred.

B.4 Resend Order Response

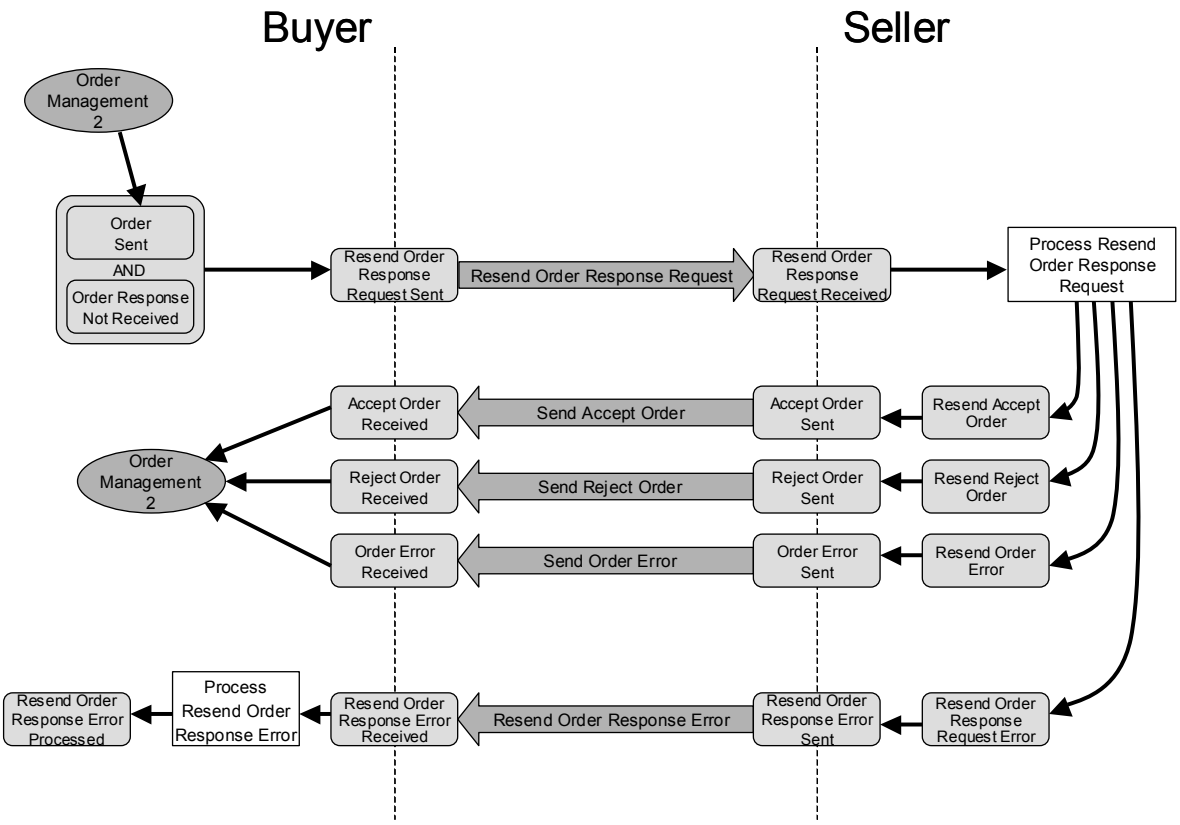


Figure 6: Resend Order Response – depends on Order Management 2

Resend Order Response is also a dependent Choreography that depends on Order Management 2. It is used to request the resending of the message sent by the seller in response to the original order.

B.5 Example XML

This section contains sample XML for the four related choreographies described above.

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- edited with XML Spy v4.4 U (http://www.xmlspy.com) by David Burdett (Commerce One) -->
<!-- Copyright Commerce One Operations Inc, (c) 2003. All rights reserved. -->
```

```
<Choreography defaultLanguage="us-en"
xmlns="com.commerceone.schemas/choreography/choreographydefinitions/choreographydefinitions.xsd"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="com.commerceone.schemas/choreography/choreographydefinitions/choreographydef
initions.xsd
C:\XML\TestSchemas\Schemas\CHOREO~1\Choreography.xsd">
 <Description>This section contains a set of choreographies for Order Management</Description>
 <!-- ROLES -->
 <Role name="Buyer">
   <Description>This role represents the purchaser of goods or services</Description>
   <!-- Order Placement States -->
   <State name="NewOrderCreated">
     <Description>Each definition of a state can have a description which can be used to
precisely explain the semantics of the state. They have been omitted from this
example.</Description>
   </State>
   <State name="OrderSent"/>
   <State name="AcceptOrderReceived"/>
   <State name="AcceptOrderCheckedOK"/>
   <State name="AcceptOrderCheckedError"/>
   <State name="AcceptOrderErrorSent"/>
   <State name="RejectOrderReceived"/>
   <State name="RejectOrderCheckedOK"/>
   <State name="RejectOrderCheckedError"/>
   <State name="RejectOrderErrorSent"/>
   <State name="OrderErrorReceived"/>
   <State name="OrderErrorMessageChecked"/>
   <!-- Order Status States -->
   <State name="OrderStatusCheckRequired"/>
   <State name="OrderStatusRequestSent"/>
   <State name="OrderStatusResponseReceived"/>
   <State name="OrderStatusResponseProcessed"/>
   <State name="OrderStatusErrorReceived"/>
   <State name="OrderStatusErrorProcessed"/>
   <!-- Resend Order Response States -->
   <State name="OrderResponseNotReceived"/>
   <State name="ResendOrderResponseRequestSent"/>
   <State name="ResendOrderResponseErrorReceived"/>
   <State name="ResendOrderResponseErrorProcessed"/>
 </Role>
 <Role name="Seller">
   <Description>This role represents the seller of goods or services</Description>
   <!-- Order Placement States -->
   <State name="OrderReceived"/>
   <State name="OrderCheckedOK"/>
   <State name="AcceptOrderSent"/>
   <State name="AcceptOrderErrorReceived"/>
   <State name="AcceptOrderErrorChecked"/>
   <State name="OrderCheckedRejected"/>
   <State name="RejectOrderSent"/>
   <State name="RejectOrderErrorReceived"/>
   <State name="RejectOrderErrorChecked"/>
   <State name="OrderCheckedError"/>
   <State name="OrderErrorSent"/>
   <!-- Order Status States -->
   <State name="OrderStatusRequestReceived"/>
   <State name="OrderStatusRequestProcessedOK"/>
   <State name="OrderStatusResponseSent"/>
   <State name="OrderStatusRequestProcessedError"/>
   <State name="OrderStatusErrorSent"/>
   <!-- Resend Order Response States -->
   <State name="ResendOrderResponseRequestReceived"/>
   <State name="ResendAcceptOrder"/>
```

```
  <State name="ResendRejectOrder"/>
  <State name="ResendOrderError"/>
  <State name="ResendOrderResponseRequestError"/>
  <State name="ResendOrderResponseErrorSent"/>
</Role>
<!-- MESSAGES-->
<!-- ORDER MANAGEMENT MESSAGES -->
<MessageFamily name="Order"
urn="rrn:org.xcbl/messagefamilies/xcblmessagefamilies/v1_0/OrderMessageFamily.xml">
  <Description>Messages in this family contain information to convey a request to purchase goods
or services</Description>
</MessageFamily>
<MessageFamily name="OrderResponse"
urn="rrn:org.xcbl/messagefamilies/xcblmessagefamilies/v1_0/OrderResponseMessageFamily.xml">
  <Description>Messages in this family contain information that is a response to a request to
purchase goods or services</Description>
</MessageFamily>
<!-- ORDER STATUS MESSAGES -->
<MessageFamily name="OrderStatusRequest"
urn="rrn:org.xcbl/messagefamilies/xcblmessagefamilies/v1_0/OrderStatusRequestMessageFamily.xml">
  <Description>Messages in this family request the status of the processing of an Order Message
sent earlier</Description>
</MessageFamily>
<MessageFamily name="OrderStatusResponse"
urn="rrn:org.xcbl/messagefamilies/xcblmessagefamilies/v1_0/OrderStatusResponseMessageFamily.xml"
>
  <Description>Messages in this family provide information on the status of the processing of an
Order Message sent earlier</Description>
</MessageFamily>
<!-- RESEND AN EARLIER REQUEST MESSAGES -->
<MessageFamily name="ResendMessageRequest"
urn="rrn:org.xcbl/messagefamilies/xcblmessagefamilies/v1_0/ResendMessageRequestMessageFamily.xml
">
  <Description>Messages in this family request the resending of a message sent
earlier</Description>
</MessageFamily>
<!-- ERROR MESSAGES -->
<MessageFamily name="ErrorMessage"
urn="rrn:org.xcbl/messagefamilies/xcblmessagefamilies/v1_0/ErrorResponseMessageFamily.xml">
  <Description>Messages in this family report errors detected in other messages which prevent
that message from being proccessed properly</Description>
</MessageFamily>
<!-- INTERACTION DEFINITIONS -->
<!-- ORDER MANAGEMENT INTERACTIONS -->
<InteractionDef name="SendOrder" fromRole="Buyer" toRole="Seller" messageFamily="Order">
  <Description>Send the order From the Buyer to the Seller</Description>
  <InteractionEndStates fromState="OrderSent" toState="OrderReceived"/>
</InteractionDef>
<InteractionDef name="SendAcceptOrder" fromRole="Seller" toRole="Buyer"
messageFamily="OrderResponse">
  <Description>The order is OK - send Order Response</Description>
  <InteractionEndStates fromState="AcceptOrderSent" toState="AcceptOrderReceived"/>
</InteractionDef>
<InteractionDef name="SendRejectOrder" fromRole="Seller" toRole="Buyer"
messageFamily="OrderResponse">
  <Description>The order was rejected - send Order Response</Description>
  <InteractionEndStates fromState="RejectOrderSent" toState="RejectOrderReceived"/>
</InteractionDef>
<InteractionDef name="SendOrderError" fromRole="Seller" toRole="Buyer"
messageFamily="ErrorMessage">
  <Description>The order was in error - send Error Message</Description>
  <InteractionEndStates fromState="OrderErrorSent" toState="OrderErrorReceived"/>
</InteractionDef>
```

```
<InteractionDef name="SendAcceptOrderError" fromRole="Buyer" toRole="Seller"
messageFamily="ErrorMessage">
  <Description>Accept Order Response in error - send Error Message</Description>
  <InteractionEndStates fromState="AcceptOrderErrorSent" toState="AcceptOrderErrorReceived"/>
</InteractionDef>
<InteractionDef name="SendRejectOrderError" fromRole="Buyer" toRole="Seller"
messageFamily="ErrorMessage">
  <Description>Reject Order Response in error - send Error Message</Description>
  <InteractionEndStates fromState="RejectOrderErrorSent" toState="RejectOrderErrorReceived"/>
</InteractionDef>
<!-- ORDER STATUS REQUEST INTERACTIONS -->
<InteractionDef name="SendOrderStatusRequest" fromRole="Buyer" toRole="Seller"
messageFamily="OrderStatusRequest">
  <InteractionEndStates fromState="OrderStatusRequestSent"
toState="OrderStatusRequestReceived"/>
</InteractionDef>
<InteractionDef name="SendOrderStatusResponse" fromRole="Seller" toRole="Buyer"
messageFamily="OrderStatusResponse">
  <InteractionEndStates fromState="OrderStatusResponseSent"
toState="OrderStatusResponseReceived"/>
</InteractionDef>
<InteractionDef name="SendOrderStatusError" fromRole="Seller" toRole="Buyer"
messageFamily="ErrorMessage">
  <InteractionEndStates fromState="OrderStatusErrorSent" toState="OrderStatusErrorReceived"/>
</InteractionDef>
<!-- RESEND ORDER RESPONSE INTERACTIONS -->
<InteractionDef name="ResendOrderResponseRequest" fromRole="Buyer" toRole="Seller"
messageFamily="ResendMessageRequest">
  <InteractionEndStates fromState="ResendOrderResponseRequestSent"
toState="ResendOrderResponseRequestReceived"/>
</InteractionDef>
<InteractionDef name="ResendOrderResponseError" fromRole="Seller" toRole="Buyer"
messageFamily="ErrorMessage">
  <InteractionEndStates fromState="ResendOrderResponseErrorSent"
toState="ResendOrderResponseErrorReceived"/>
</InteractionDef>
<!-- CHOREOGRAPHY DEFINITIONS -->
<!-- ORDER MANAGEMENT 1 -->
<ChoreographyDefinition name="OrderManagementChoreography1"
urn="rrn:org.xcbl:choreographies/ordermanagement/v1_0/ordermanagementchoreography1.xml">
  <Description>In this Choreography Definition, a Buyer sends an Order to a Seller. The Seller
returns an Error Message, if the Order cannot be processed</Description>
  <StartEndStates>
    <Start state="NewOrderCreated"/>
    <ConditionalEnd state="OrderSent"/>
    <End state="OrderCheckedOK"/>
    <End state="OrderCheckedRejected"/>
    <End state="OrderErrorSent"/>
    <End state="OrderErrorMessageChecked"/>
  </StartEndStates>
  <Interaction name="SendOrder">
    <Description>Send the order to the seller</Description>
    <PreCondition condition="NewOrderCreated"/>
  </Interaction>
  <Process name="CheckOrder" role="Seller">
    <Description>The seller checks the order.</Description>
    <PreCondition condition="OrderReceived"/>
    <ProcessEndState state="OrderCheckedOK"/>
    <ProcessEndState state="OrderCheckedRejected"/>
    <ProcessEndState state="OrderCheckedError"/>
  </Process>
  <Interaction name="SendOrderError">
    <Description>The order was in error - send an error</Description>
```

```
    <PreCondition condition="OrderCheckedError"/>
  </Interaction>
  <Process name="ProcessOrderErrorMessage" role="Buyer">
    <Description>Buyer Processes Order Error Message</Description>
    <PreCondition condition="OrderErrorReceived"/>
    <ProcessEndState state="OrderErrorMessageChecked"/>
  </Process>
</ChoreographyDefinition>
<!-- ORDER MANAGEMENT 2 -->
<ChoreographyDefinition name="OrderManagementChoreography2"
urn="rrn:org.xcbl:choreographies/ordermanagement/v1_0/ordermanagementchoreography1.xml">
  <Description>This choreography allows a Buyer to send an Order message to a Seller and receive
either an Order Response Message or an Error Message in return. If the Order Response Message is
in error, then the Buyer sends an Error Message to the Seller.</Description>
  <ExtendsChoreography
urn="rrn:org.xcbl:choreographies/ordermanagement/v1_0/ordermanagementchoreography1.xml"/>
  <StartEndStates>
    <Start state="NewOrderCreated"/>
    <ConditionalEnd state="AcceptOrderSent"/>
    <ConditionalEnd state="RejectOrderSent"/>
    <End state="OrderErrorSent"/>
    <End state="OrderCheckedOK"/>
    <End state="OrderCheckedRejected"/>
    <End state="RejectOrderErrorSent"/>
    <End state="AcceptOrderErrorSent"/>
    <End state="RejectOrderErrorChecked"/>
    <End state="AcceptOrderErrorChecked"/>
  </StartEndStates>
  <Interaction name="SendAcceptOrder">
    <Description>Accept the Order</Description>
    <PreCondition condition="OrderCheckedOK"/>
  </Interaction>
  <Interaction name="SendRejectOrder">
    <Description>Reject the Order</Description>
    <PreCondition condition="OrderCheckedRejected"/>
  </Interaction>
  <Process name="CheckAcceptOrder" role="Buyer">
    <Description>Buyer Checks Accept Order Response</Description>
    <PreCondition condition="AcceptOrderReceived"/>
    <ProcessEndState state="AcceptOrderCheckedOK"/>
    <ProcessEndState state="AcceptOrderCheckedError"/>
  </Process>
  <Process name="CheckRejectOrder" role="Buyer">
    <Description>Buyer Checks Reject Order Response</Description>
    <PreCondition condition="RejectOrderReceived"/>
    <ProcessEndState state="RejectOrderCheckedOK"/>
    <ProcessEndState state="RejectOrderCheckedError"/>
  </Process>
  <Interaction name="SendAcceptOrderError">
    <PreCondition condition="CheckAcceptOrderError"/>
  </Interaction>
  <Process name="ProcessAcceptOrderError" role="Seller">
    <Description>Seller processes the Send Accept Order Error Message</Description>
    <PreCondition condition="AcceptOrderErrorReceived"/>
    <ProcessEndState state="AcceptOrderErrorChecked"/>
  </Process>
  <Interaction name="SendRejectOrderError">
    <PreCondition condition="CheckRejectOrderError"/>
  </Interaction>
  <Process name="ProcessRejectOrderError" role="Seller">
    <Description>Seller processes the Send Reject Order Error Message</Description>
    <PreCondition condition="RejectOrderErrorReceived"/>
    <ProcessEndState state="RejectOrderErrorChecked"/>
```

```
</ChoreographyDefinition>
<!-- CHECK ORDER STATUS-->
<ChoreographyDefinition name="OrderManagementCheckOrderStatus"
urn="rrn:org.xcbl:choreographies/ordermanagement/v1_0/ordermanagementcheckorderstatus.xml">
  <Description>This choreography allows a Buyer to check on the status of the processing of an
order sent to the seller earlier.</Description>
  <DependsOnChoreography
urn="rrn:org.xcbl:choreographies/ordermanagement/v1_0/ordermanagementchoreography1.xml"/>
  <StartEndStates>
    <Start state="OrderStatusCheckRequired"/>
    <End state="OrderStatusResponseSent"/>
    <End state="OrderStatusErrorSent"/>
    <End state="OrderStatusResponseProcessed"/>
    <End state="OrderStatusErrorProcessed"/>
  </StartEndStates>
  <Interaction name="SendOrderStatusRequest">
    <Description>The sending of an Order Status is also dependent on an Order being
sent.</Description>
    <PreCondition condition="OrderSent and OrderStatusCheckRequired"/>
  </Interaction>
  <Process name="ProcessOrderStatusRequest" role="Seller">
    <PreCondition condition="OrderStatusRequestReceived"/>
    <ProcessEndState state="OrderStatusRequestProcessedOK"/>
    <ProcessEndState state="OrderStatusRequestProcessedError"/>
  </Process>
  <Interaction name="SendOrderStatusResponse">
    <PreCondition condition="OrderStatusRequestProcessedOK"/>
  </Interaction>
  <Interaction name="SendOrderStatusError">
    <PreCondition condition="OrderStatusRequestProcessedError"/>
  </Interaction>
  <Process name="ProcessOrderStatusResponse" role="Buyer">
    <PreCondition condition="OrderStatusResponseReceived"/>
    <ProcessEndState state="OrderStatusResponseProcessed"/>
  </Process>
  <Process name="ProcessOrderStatusError" role="Buyer">
    <PreCondition condition="OrderStatusErrorReceived"/>
    <ProcessEndState state="OrderStatusErrorProcessed"/>
  </Process>
</ChoreographyDefinition>
<!-- RESEND ORDER RESPONSE-->
<ChoreographyDefinition
urn="rrn:org.xcbl:choreographies/ordermanagement/v1_0/ordermanagementrecovery.xml"
name="OrderManagementRecovery">
  <Description>This choreography allows a Buyer to recover the processing of an order if a
message is not received. It includes: checking the state of the processing at the seller
followed by a request to resend the latest order information.</Description>

  <DependsOnChoreography
urn="rrn:org.xcbl:choreographies/ordermanagement/v1_0/ordermanagementchoreography2.xml"/>
  <StartEndStates>
    <Start state="OrderResponseNotReceived"/>
    <End state="AcceptOrderSent"/>
    <End state="RejectOrderSent"/>
    <End state="OrderErrorSent"/>
    <End state="ResendOrderResponseErrorSent"/>
    <End state="ResendOrderResponseErrorProcessed"/>
  </StartEndStates>
  <Interaction name="ResendOrderResponseRequest">
    <Description>This interaction is dependent on an order being sent with no message received
in response</Description>
    <PreCondition condition="OrderSent and OrderResponseNotReceived"/>
```

```
  </Interaction>
  <Process name="ProcessResendOrderResponseRequest" role="Seller">
    <PreCondition condition="ResendOrderResponseRequestReceived"/>
    <ProcessEndState state="ResendAcceptOrder"/>
    <ProcessEndState state="ResendRejectOrder"/>
    <ProcessEndState state="ResendOrderError"/>
    <ProcessEndState state="ResendOrderResponseRequestError"/>
  </Process>
  <Interaction name="SendAcceptOrder">
    <PreCondition condition="ResendAcceptOrder"/>
  </Interaction>
  <Interaction name="SendRejectOrder">
    <PreCondition condition="ResendRejectOrder"/>
  </Interaction>
  <Interaction name="SendOrderError">
    <PreCondition condition="ResendOrderError"/>
  </Interaction>
  <Interaction name="ResendOrderResponseError">
    <PreCondition condition="ResendOrderResponseRequestError"/>
  </Interaction>
 </ChoreographyDefinition>
</Choreography>
```