WS Choreography Model Overview, Version 0-1

Editor's Draft, 4 December 2003

This version:
TBD
Latest version:
TBD
Previous Version:
Not Applicable

Editors (alphabetically):
David Burdett, Commerce One, mailto:david.Burdett@commerceone.com
Nickolaos Kavantzas, Oracle, Oracle mailto:nickolas.kavantzas@oracle.com

This document is available in other format(s):

Copyright

Abstract

It's purpose is to provide an information model that describes the data and the relationships between them that is needed to define a choreography that describes the sequence and conditions in which the data exchanged between two or more participants in order to meet some useful purpose.

Status of this Document

This is the first version of the WS Choreography Model Overview paper. It has no official status within the W3C.

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

Table of Contents

1 Introduction

1.1 Purpose

Business or other activities that involve multiple different organizations or independent processes that use Web service technology to exchange information can only be successful if they are properly coordinated. This means that the sender and receiver of a message know and agree in advance:

- The format and structure of the (SOAP) messages that are exchanged, and
- The sequence and conditions in which the messages are exchanged.

WSDL and its extensions provide a mechanism by which the first objective is realized, however, it does not define the sequence and conditions, or *choreography*, in which messages are exchanged.

To solve this problem, a shared common or “global” definition of the sequence and conditions in which messages are exchanged is produced that describes the observable complementary behavior of all the participants involved. Each participant can then use the definition to build and test solutions that conform to the global definition.

The main advantage of a global definition approach is that it separates the process being followed by an individual business or system within a “domain of control” from the definition of the sequence in which each business or system exchanges information with others. This means that, as long as the “observable” sequence does not change, the rules and logic followed within the domain of control can change at will.

The purpose of this paper is to describe an information model or “meta model” for a Choreography Definition Language that identifies the information and structures required to build a “global” definition.

1.2 Goals

Some additional goals of this model of a choreography definition language are to permit:

- *Reusability*. The same choreography definition is usable by different participants operating in different contexts (industry, locale, etc) with different software (e.g. application software) and different message formats and standards
- *Cooperative*. Choreographies define the sequence of exchanging messages between two (or more) independent participants or processes by describing how they should cooperate
- *Multi-Party*. Choreographies can be defined involving any number of participants or processes
- *Semantics*. Choreographies can include human-readable documentation and semantics for all the components in the choreography.

- *Composability*. Existing choreographies can be combined to form new choreographies that may be reused in different contexts
- *Modular.* Choreographies can be defined using an "import" facility that allows a choreography to be created from components contained in several different choreographies
- *Information Driven*. Choreographies describe how participants that take part in choreographies maintain where they are in the choreography by recording the state changes caused by exchanges of information and their reactions to them
- *Information Alignment*. Choreographies allow the participants that take part in choreographies to communicate and synchronize their states and the information they share
- *Transactionality*. The processes or participants that take part in a choreography can work in a “transactional” way with the ability to specify how transactions are compensated
- *Exception Handling*. Choreographies can define how exceptional or unusual conditions that occur whilst the choreography is performed are handled
- *Design Time Verification*. A developer of a business process can use the Choreography Definition, on their own to:
  - Generate a behavioral interface that conforms to a BPEL definition that describes the sequence and conditions in which one of the participants in a choreography sends and receives messages
  - Verify that a BPEL definition conforms to behavior defined by in a Choreography Definition
- *Run Time Verification*. The performance of a choreography can be verified at run time against the Choreography Definition to ensure that it is being followed correctly. If errors are found then the choreography can specify the action that should be taken
- *Compatibility with other Specifications*. The specifications will work alongside and complement other specifications such as the WS Reliability, WS WS Composite Application Framework (WSCAF), WS Security (WSS), WS Business Process Execution Language (WSBPEL) etc.

1.3 Document Scope

This model focuses on describing the different types of information required to define a Choreography. It does not provide an XML representation of that information nor does it describe in any detail the operational semantics of how such a representation could or should be used.

This paper identifies several open issues highlighted like this. These are a non-exhaustive list of topics, ideas or problems where the authors think that more thought is needed.

2 Abstract, Portable and Concrete Choreographies

One of the key goals of this model is to enable Choreography reuse. Global definitions of a choreography facilitate this especially if choreographies are defined with varying degrees of abstraction. Although more could be defined, this model identifies and supports three different levels of abstraction in which choreographies can usefully be defined and used.

2.1 Abstract Choreography

The first is a highly abstract choreography that defines:

- The types of information that is exchanged, for example an order sent between a buyer and a seller
- The sequence and conditions under which the information is sent.

However, it does not define:

- The physical structure of the information that is exchanged, for example there are no definitions of the XML documents, SOAP messages, WSDL port types and operations, URLs etc that are to be used
- How the different conditions that are used to control the sequence of exchanging information are determined
- Where the messages in the choreography should be sent e.g. to a URL
- How the messages are to be secured (if at all) and whether or not the messages are to be sent reliably.

Although abstract, this approach will be useful for defining generally accepted or “canonical” definitions for very common processes, such as placing an order. Definitions of theses types of choreography would best be carried out by international standards organizations that have a cross-industry, multi-geographic responsibility.

2.2 Portable Choreography

Clearly, the development of these abstract choreographies will take some time to complete, so the second type of choreography to define is a “portable” choreography. In this type of choreography definition the definitions in an Abstract Choreography are extended with:

- Detailed definitions of the physical structure of the information that is exchanged including the WSDL port types and operations
- Details of the technology to be used, for example, how to secure the messages and send them reliably
- Rules that express, as far as possible, the conditions that are used to control the sequence of exchange of information, in terms of, for example XPath expressions that reference data in the messages

However they do not specify the URLs to which the messages are sent nor, for example, the digital certificates used to secure them. This means that an organization should be able to design and build a solution that conforms, in detail, to the rules of the choreography, and only require limited additional information at run time to determine where messages should be sent. As a result realizing interoperability should be much easier.

This “portable” type of choreography is targeted more at vertical industry organizations, such as RosettaNet, that want to define rules for collaboration between the members of their industry and simplify, as far as possible, the implementation and integration process.

2.3 Concrete Choreographies

The final type of choreography, is a Concrete Choreography, where all the details are specified that are required to send a message. This extends the definition in a Portable Choreography to include information about the “endpoints”. This can include information such as:

- The URLs that are the destinations of the messages that are sent, and
- Other “endpoint” specific rules such as digital certificates to be used for securing messages.

These types of choreographies are probably most applicable where two or more participants want to specify how they will cooperate and there is little or no need for other organizations to follow the same process.

2.4 Relationship between Choreography Types

The table below summarizes the three different types of choreographies.

	Abstract	**Portable**	**Concrete**
Types of Messages	Identified	Identified	Identified
Message Structure	Not Defined	Defined	Defined
Conditions	Identified	Identified	Identified
Condition evaluation rules	Not defined	Defined as far as possible	Defined as far as possible
Technology used	Not defined	Defined	Defined
Message Endpoint Data	Not defined	Not Defined	Defined

The model described in this paper allows an Abstract Choreography to be extended to become a Portable Choreography and a Portable Choreography to be extended to become a Concrete Choreography.

The model also allows each different type of Choreography to be defined directly. This means that:

- A Portable Choreography can be defined without first defining the Abstract Choreography
- A Concrete Choreography can be defined without defining an Abstract or Portable Choreography.

3 Model Description

The following diagram is the full model of all the entities (without attributes).

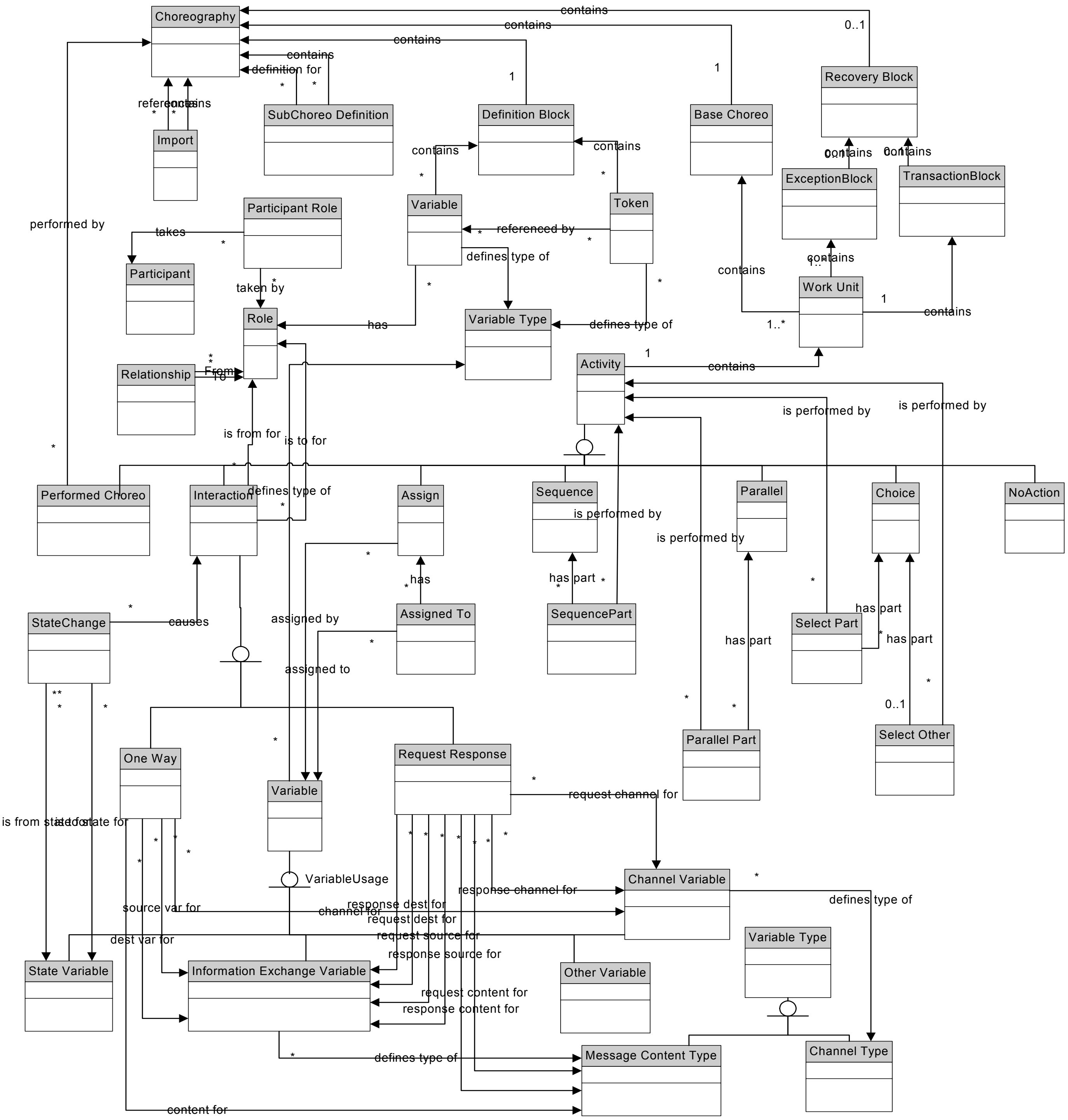

Figure 1: Full Model

The rest of this Model Description section describes the following subsets of the model (including attributes).

- *Participants, Roles and Relationships*. In a Choreography information is always exchanged between Participants, such as a Business or Organization acting in one or more *Roles*, for example Buyer or Seller as part of a Relationship, for example purchasing goods.
- *Choreography Structure*. This section describes the major components of a Choreography at a high level
- *Choreography Composition and Import*. This explains how one Choreography can be created by performing other, pre-existing choreographies and importing content from other choreographies.
- *Types, Variables and Tokens*. *Variables* contain information about objects in the choreography such as the messages exchanged or the state of the *Roles* involved. *Tokens* are aliases that can be used to reference parts of a *Variable*. Both *Variables* and *Tokens* have *Types* that define the structure of what the *Variable* or *Token* contains.
- *Interactions*. These are the basic building blocks of the Choreography which result in the sending of messages between Roles in either a “one-way” or “request-response” message pattern
- *Activities and Control Structures*. Activities are the lowest level components of the Choreography that do the actual work. Control Structures combine activities with other Control Structures in a nested structure to express the sequence and conditions in which the messages in the choreography are exchanged
- *Choreography Exceptions and Transactions*. *Choreography Exceptions* describe how to specify what additional Interactions should occur when a Choreography behaves in an abnormal way. *Choreography Transactions* describes how to specify what additional Interactions should occur to reverse the effect of an earlier completed choreography
- *Semantics*. Semantics allow the creation of descriptions that can record the semantic definitions of almost every single component in the model.

3.1 Roles, Participants and Relationships

In a Choreography information is always exchanged between Participants, such as a Business or Organization acting in one or more Roles, for example Buyer or Seller as part of a Relationship, for example purchasing goods.

The diagram below shows the model for Participants, Roles and Relationships.

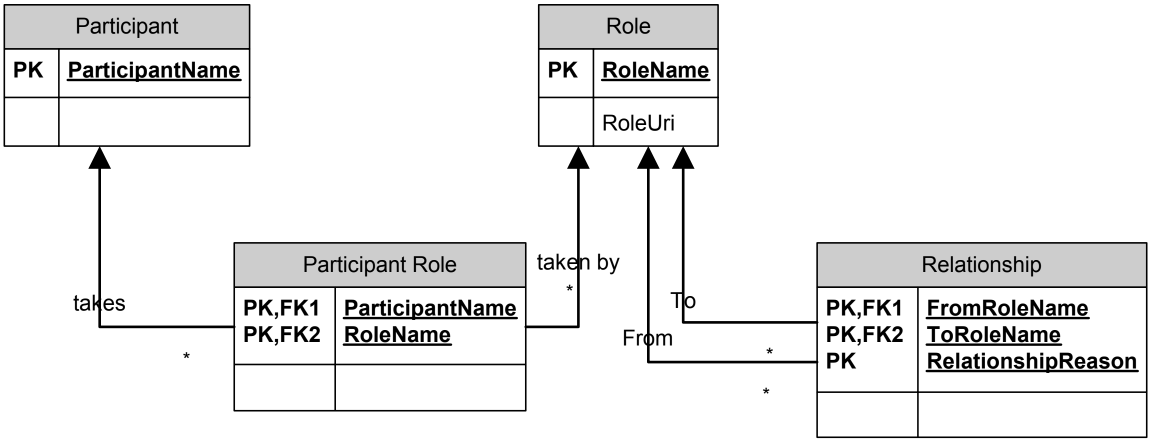

Figure 2: Model for Participants, Roles and Relationships

3.1.1 Roles

A Role identifies a set of related behaviors, for example the Buyer role is associated with purchasing of goods or services and the Supplier role is associated with providing those goods or services for a fee.

3.1.2 Participants

A Participant identifies a set of related Roles, for example a Commercial Organization could take both a Buyer Role when purchasing goods and a Seller role when selling them.

3.1.3 Relationship

A Relationship is the association of two Roles for a purpose. A relationship represents the possible ways in which two roles can interact. For example the Relationships between a Buyer and a Seller could include:

- A “Purchasing” Relationship, for the initial procurement of goods or services, and
- A “Customer Management” Relationship to allow the Supplier to provide service and support after the goods have been purchased or the service provided.

Although Relationships are always between two Roles, Choreographies involving more than two Roles are possible. For example if the purchase of goods involved a third-party Shipper contracted by the Supplier to deliver the Supplier’s goods, then, in addition to the Purchasing and Customer Management relationships described above, the following relationships might exist:

- A “Logistics Provider” relationship between the Supplier and the Shipper, and
- A “Goods Delivery” relationship between the Buyer and the Shipper.

3.2 Choreography Structure

The diagram below shows the model for a Choreography Definition:

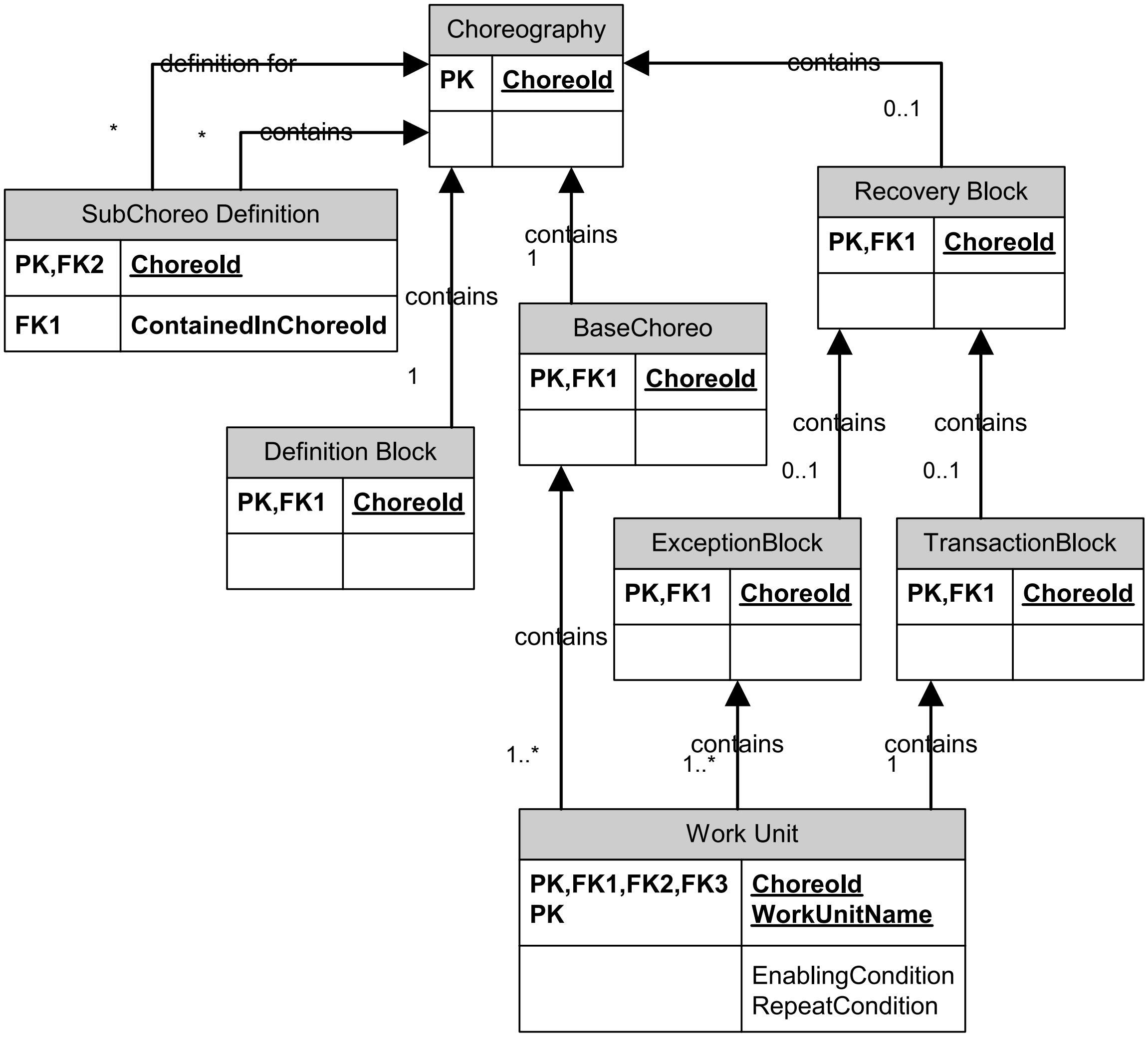

Figure 3: Model for Choreography Structure

A *Choreography Definition* defines the information required by the choreography and sequence in which it is exchanged. It contains the following:

- Zero or More "sub" Choreography Definitions which define Choreographies that can be performed by the Choreography being defined
- A *Definition Block* that contains set of *Variable Definitions* and *Token Definitions* that define information about objects used by the choreography
- The actual *Choreography* that in turn contains:
  - A required *Base Choreography* part, that defines the normal sequence of information exchanges that should occur

  - An optional *Exception Block*, that contains the sequence of information exchanges that are followed when some exceptional or unusual circumstance has occurred while the Choreography was being performed, and
  - An optional *Transaction Block* which, if present can make the Choreography “transactional” in that it contains information exchanges that are followed when the effects of the choreography need to be *Compensated*

- One or more *Work Units*, within the *Base Choreography, Exception Block* or *Transaction Block* that do the actual useful work within the Choreography in terms of exchanging messages and other information between the Participants. Each *Work Unit* contains a single *Activity* that is performed whenever an optional enabling condition on the *Work Unit*, called a *Guard*, is true.

Issue CS-01. For the XML need to work out how Namespaces, etc are handled.

3.3 Choreography Composition and Import

Choreographies can be combined and built from other Choreographies as illustrated by the diagram below.

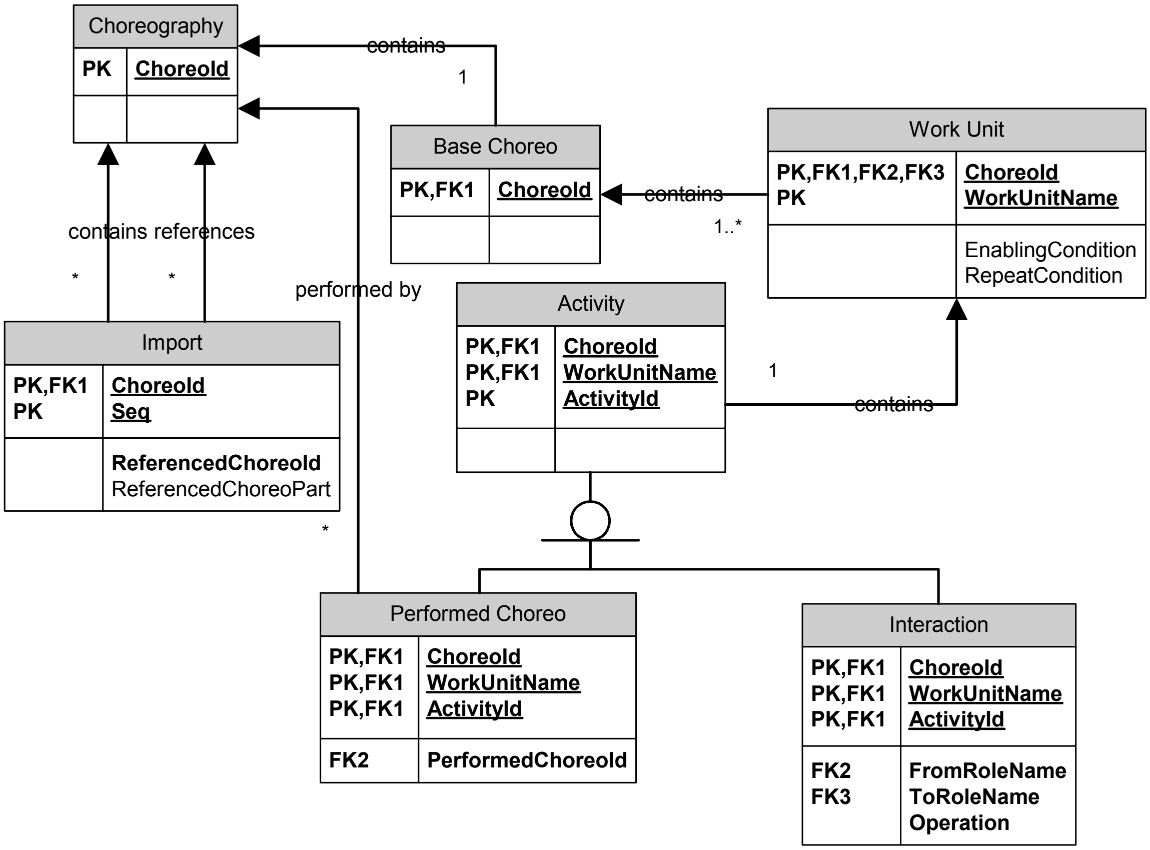

Figure 4: Model for Choreography Composition and Import

3.3.1 Choreography Composition

Choreography Composition is the creation of new Choreographies by reusing existing choreography definitions. For example if two separate Choreographies were defined as follows:

- A Request for Quote (RFQ) Choreography that involved a Buyer role sending a request for a quotation for goods and services to a Supplier to which the Supplier responding with either a “Quotation” or a “Decline to Quote” message, and
- An Order Placement Choreography where the Buyer placed and order for goods or services and the Supplier either accepted the order or rejected it.

You could then create a new “Quote and Order” Choreography by reusing the two where the RFQ choreography was executed first, and then, depending on the outcome of the RFQ Choreography, the order was placed using the Order Placement Choreography.

In this case the new choreography is “composed” out of the two previously defined choreographies. These choreographies may be specified either:

- *Locally*, i.e. they are included, as a *Sub Choreography,* in the same choreography definition as the choreography that performed them, or
- *Globally,* i.e. they are specified in a separate choreography definition that is defined elsewhere.

Using this approach, Choreographies can be recursively combined to support choreographies of any required complexity allowing more flexibility as Choreographies defined elsewhere can be reused.

3.3.2 Import Statements

An *Import* statement can contain references to a complete Choreography or part of a Choreography.

Import statements must be interpreted in the sequence they occur.

When the *Import* statement contains references to variables or other data that have the same identity, then the content of the later *Import* statement replaces the same content referenced by the earlier *Import* statement.

This means, for example, that if an initial Choreography definition referenced by an *Import* statement contained variables, etc, that were defined in an Abstract way, then the replacement definition could either be Portable or Concrete.

It also enables one Choreography definition to effectively be “cloned” by replacing the definitions for some or all of its variables.

Issue CCI-01. How are definitions identified as being the same and therefore should be overridden?

Issue CCI-02.Import statements need to apply at potentially other points within the choreography apart from the top level?

3.4 Types, Variables and Tokens

Variables contain information about objects in the choreography such as the messages exchanged or the state of the *Roles* involved. *Tokens* are aliases that can be used to reference parts of a *Variable*. Both *Variables* and *Tokens* have *Types* that define the structure of what the *Variable* or *Token* contains.

The diagram below shows the model for Types, Variables and Tokens:

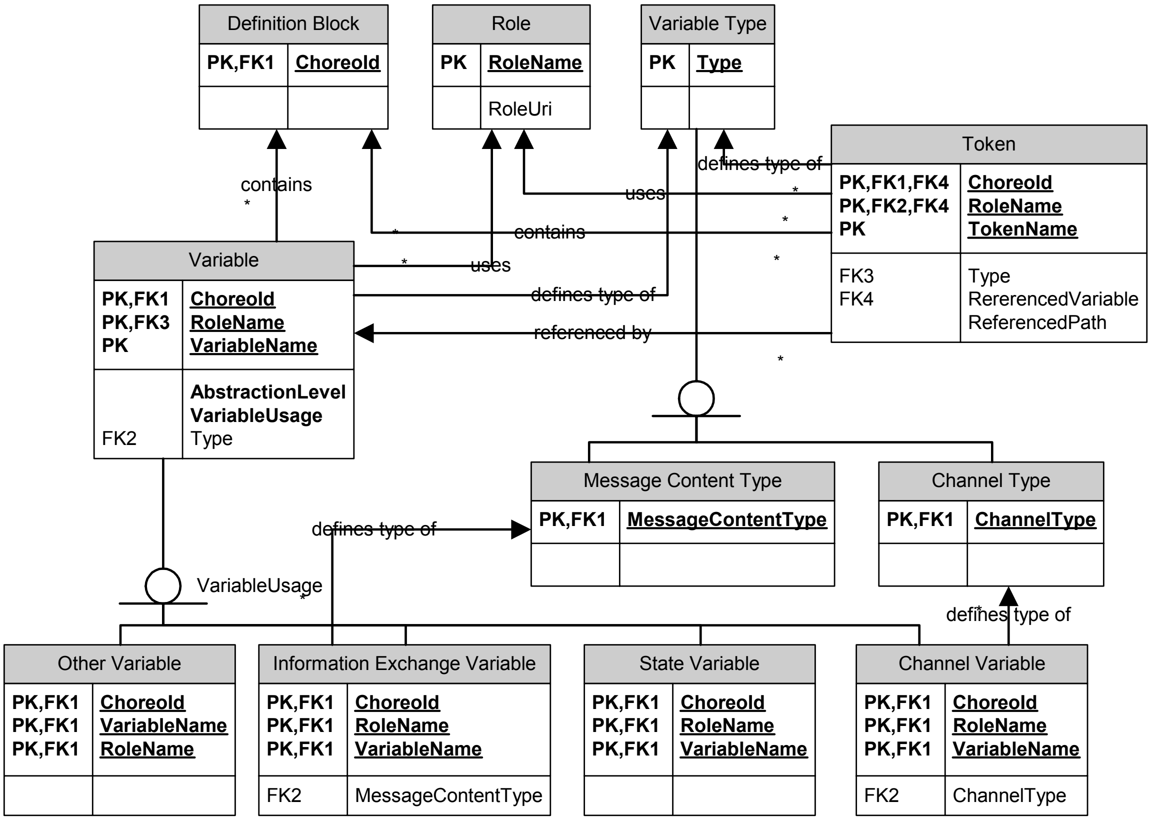

Figure 5: Model for Types, Variables and Tokens

3.4.1 Types

3.4.1.1 Variable Types

Variable Types describe the type of information that is being captured within a Variable at a Role. The type of information that is referenced will vary depending on the type of the Choreography and the type of information that the variable contains.

Choreography Type	Variable Type
Abstract	In an Abstract Choreography, the Variable Type is described by: ▪ A unique identifier, e.g. a URI, that identifies the variable

Choreography Type	**Variable Type**
	type and ▪ A semantic definition that explains the purpose of the variable type and outlines its content. No detail is provided of the actual type, e.g. XSD definitions
Portable	In a Portable Choreography the Variable Type extends the Abstract Variable Type by defining its XML Schema Type. Note that this may be simple or complex depending on the need.
Concrete	In a Concrete Choreography, Variable Type is defined in the same way as for a Portable Choreography

Issue TVT-01 should this be extended to include other items such as SOAP headers, security, etc when the Variable is describing a Message?

3.4.1.2 Channel Types

A Channel identifies where and how to send information to the To Role. Additionally, it identifies what is the allowed Channel information that can be passed to the To Role and the usage of a Channel within a participant.

The content varies depending on the type of the choreography:

Choreography Type	**Channel**
Abstract	In an Abstract Choreography, the Channel Type is described by: ▪ A unique identifier, e.g. a URI that identifies the Channel Type within the Role ▪ A semantic definition, that describes the type of channel information that the Channel can accept. Including: o What channel information can be passed using this channel type o How a channel should be used
Portable	In a Portable Choreography, the abstract Channel Type is extended by identifying: ▪ One or more WSDL Service Interfaces that collectively implement the channel type. ▪ How Correlation of the messages sent using the Channel Type is to be done
Concrete	Channel Types in a Concrete Choreography are defined in the same way as for a Portable Choreography.

3.4.2 Variables

Variables capture information about objects in a Choreography. They have the following usages as defined by the *Variable Usage*:

- *Information Exchange Variables* that contain information such as an Order that is used to:
  - Populate the content of a message to be sent, or
  - Populated as a result of a message received
- *State Variables* that contain information about the State of a Role as a result of information exchanged. For example:
  - When a Buyer sends an order to a Seller, the Buyer could have a *State Variable* called “OrderState” set to a value of “OrderSent” and once the message was received by the Seller, the Seller could have an *State Variable* called “OrderState” set to a value of “OrderReceived”. Note that the variable “OrderState” at the Buyer is a different variable to the “OrderState” at the Seller
  - Once an order is received, then it might be validated and checked for acceptability in other ways that affect how the choreography is performed. This could require additional states to be defined for “Order State”, such as: “OrderError”, which means an error was detected that stops processing of the message, “OrderAccepted”, which means that there were no problems with the Order and it can be processed, and “OrderRejected”, which means, although there were no errors, it cannot be processed, e.g. because a credit check failed.
- *Channel Variables* that contain information that describes how and where a message is sent to a Role. For example, a Channel Variable could contain information such as the URL to which the message should be sent, the policies that are to be applied, such as security, whether or not reliable messaging is to be used, etc.
- *Other Variables* including
  - *Locally Defined Variables* that contain information created and changed locally by a Role. They can be Information Exchange, State or Channel Variables as well as variables of other types. For example “Maximum Order Amount” could be data created by a seller that is used together with an actual order amount from an Order received to control the flow of the choreography. In this case how Maximum Order Amount is calculated and its value would not be known by the other Roles
  - *Common Variables* that contain information that is common knowledge to two or more Roles, e.g. “OrderResponseTime” which is the time in hours in which a response to an Order must be sent

The value of Variables can be:

- Known by all the roles prior to the start of the choreography
- Assigned by one role and optionally communicated to other roles
- Assigned as a result of an interaction
- Assigned by one role from other information
- Used to determine the decisions and actions to be taken in a Choreography.

The way Variables are defined will vary depending on the type of choreography.

Choreography Type	Variables
Abstract	In an abstract choreography, variables are described by: ▪ An Role name that identifies the role within which the variable is known ▪ A name that identifies the variable, that is unique within the Role within the Choreography Definition ▪ A semantic definition, that describes what the variable means
Portable	In a portable choreography, the abstract definition of the Variables is extended to include a Variable Type, which define what type of information the variable contains
Concrete	Variables in a Concrete Choreography are defined in the same way as for a Portable Choreography.

Issue TVT-02. How could (or should) we combine variables of the form , e.g. “Account Balance + Order Amount” so that it could be compared with “Credit Limit”

3.4.2.1 Variables and Abstract/Concrete Choreographies

Defining Variables to hold information about the objects in a Choreography means that:

- Variables contain all the information about a Choreography that can change from implementation to implementation
- The definition of the sequence and conditions in which information is exchanged is independent of how those information exchanges are actually implemented
- As new methods are developed for defining interfaces, messages, as well as other Web Services standards, only the way the variables are defined should need to change. The essence of the choreography, i.e. the basic definition of the sequence and conditions in which information is exchanged, remains the same.

In addition the *Import* statement also allows definitions in one choreography, to be over-ridden by other, replacement definitions. This means that:

- The same choreography can be reused in different contexts with different interfaces, message types and varying levels of detail as required
- The *Abstraction Level* of the variables can change as required from abstract through to concrete
- The definitions of the variables in an "abstract" choreography can be used as a checklist to validate that any replacement definitions at either the Portable or Concrete levels form a complete list.

3.4.3 Tokens

A Token is an alias for a piece of data in a variable or message that needs to be used by a Choreography. Tokens differ from Variables in that Variables contain values whereas Tokens contain information that defines how to access the piece of the data that is relevant.

For example a Token for "Order Amount" within an Order business could be an alias for an expression that pointed to the Order Amount element within an XML document. This could then be used as part of a condition that controls the flow of a choreography, for example "Order Amount > $1000"

All tokens may have a type, for example, an Order Amount would be of type amount, Order Id could be alphanumeric and counter an integer.

The way these tokens are defined will vary depending on the type of choreography.

Choreography Type	**Tokens**
Abstract	In an abstract choreography, tokens are described by: ▪ A unique identifier, e.g. a URI that identifies the token ▪ A semantic definition, that describes what the token means However Abstract tokens do not have a type.
Portable	In a portable choreography, a token extends an Abstract definition of a token by defining: ▪ Its type, e.g. by giving it an XML Schema type ▪ A reference to the location of the item, for example using an XML Path expression
Concrete	Tokens in a Concrete Choreography are defined in the same way as for a Portable Choreography.

3.5 Interactions

The diagram below shows the model for *Interactions*.

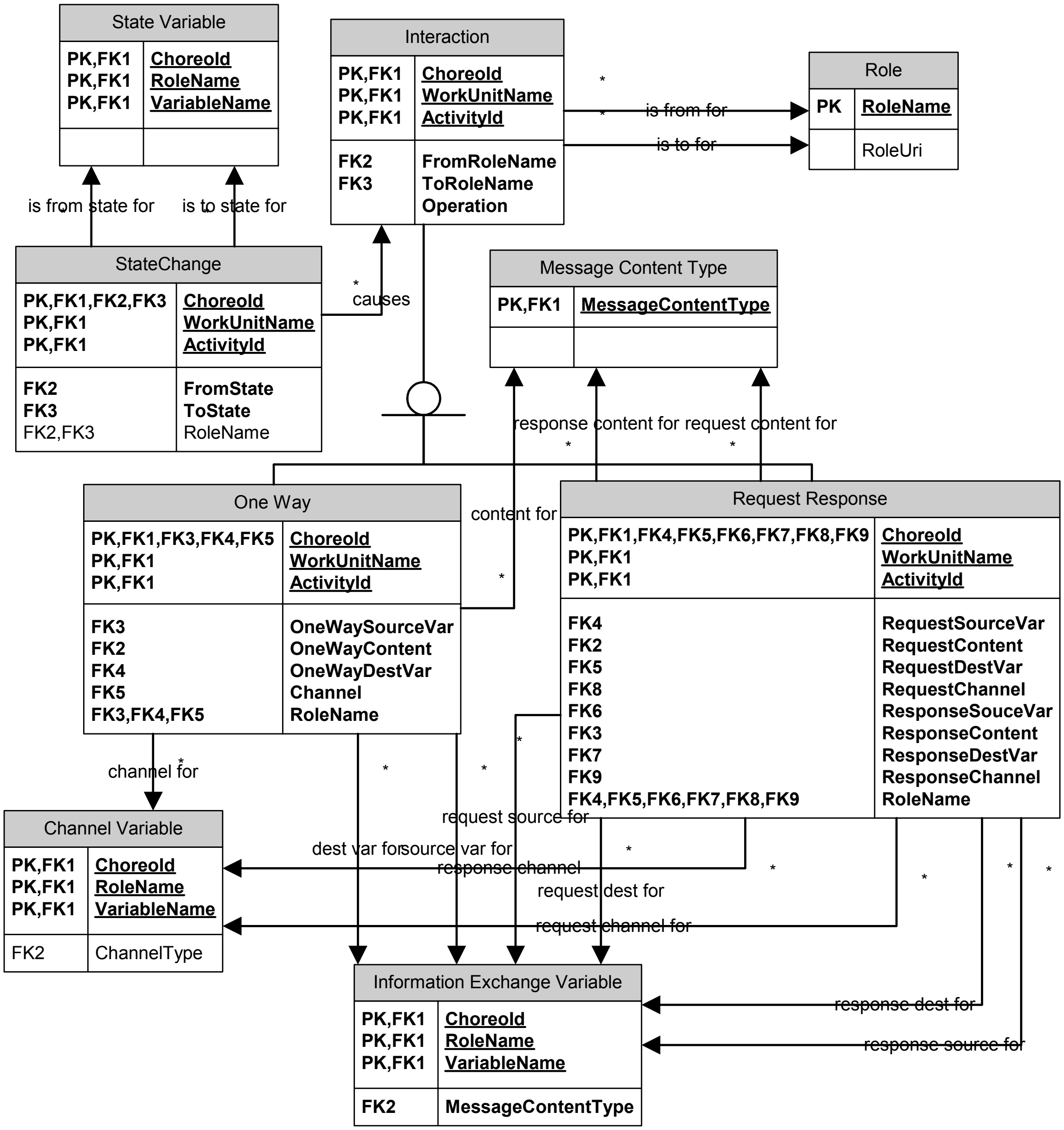

Figure 6: Model for Interactions

An *Interaction* always involves the exchange of information between two Roles in a Relationship that conform to a Message Exchange Pattern as defined by WSDL 1.2. This means an Interaction can be one of two types:

- A *One-Way Interaction* that involves the sending of single message,
- A *Request-Response Interaction* when two messages are exchanged.

An Interaction also contains “references” to:

- The *From Role* and *To Role* that are involved
- The *Message Content Type* that is being exchanged
- The *Information Exchange Variables* at the From Role and To Role that are the source and destination for the *Message Content*
- The *Channel Variable* that specifies the interface and other data that describe where and how the message is to be sent
- The *Operation* that specifies what the recipient of the message should do with the message when it is received
- A list of potential *States Changes* that can occur and may be aligned at the *From Role* and the *To Role* as a result of carrying out the Interaction.

Each of these is described below.

Issue I-01. The model diagram does not describe how error responses are handled

3.5.1 Interaction Roles

Interactions always have a “direction” in that there is a *From Role* that sends the original message and a *To Role* that receives the message. In the case of a request/response MEP, the “To Role” will also send a response message back to the “From Role”.

3.5.2 Interaction Message Content

Message Content identifies the type of information that is exchanged between the roles and the *Information Exchange Variables* used as follows:

- *One Way From Message* is the variable that is the source for a One-Way Message at the *From Role*
- *One Way To Message* is the variable that is the destination for a One-Way Message at the *To Role*
- *Request From Message* is the variable that is the source for Request Message at the *From Role*
- *Request To Message* is the variable that is the destination for Request Message at the *To Role*
- *Response To Message* is the variable that is the source for Response Message at the *To Role*
- *Response From Message* is the variable that is the destination for Response Message at the *From Role*

The type of information that is referenced will vary depending on the type of the Choreography.

Choreography Type	Message Content
Abstract	In an Abstract Choreography, the message content that is exchanged is described by: ▪ A unique identifier, e.g. a URI, that identifies the message content and ▪ A semantic definition that explains the purpose of the message and outlines its content. No detail is provided of the actual message content, e.g. XSD definitions
Portable	In a Portable Choreography, the Abstract definition of Message Content is extended to include a WSDL Message Type or an XSD element type
Concrete	In a Concrete Choreography, Message Content is defined in the same way as for a Portable Choreography

Issue I-02. Should Portable Choreography message content be extended to include other items such as SOAP headers, security, etc or should this be included in the Channel?

3.5.3 Interaction Channel Variables

A Channel Variable contains information on where and how to send information to a specific instance of the To Role. This is because Concrete Channel information plus Correlation information about a Choreography contains sufficient information to identify how to send messages to a specific instance of a process.

Additionally, Channel Variable information can be passed within Message Content. This allows the destination for messages in a choreography to be determined dynamically.

For example, a Buyer could specify Channel information to be used for sending delivery information. The Buyer could then send the Channel information to the Seller who then forwards it to the Shipper. The Shipper could then send delivery information directly to the Buyer using the Channel Information originally supplied by the Buyer.

The content varies depending on the type of the choreography.

Choreography Type	Channel
Abstract	In an Abstract Choreography, the channel is described by: ▪ A unique identifier, e.g. a URI that identifies the Channel

Choreography Type	Channel
	within the Role ▪ A semantic definition, that describes the type of channel information that the Channel can accept
Portable	In a Portable Choreography, the abstract channel is extended by identifying its Channel Type, which defines what type of information the variable contains.
Concrete	In a concrete choreography, the channel extends a portable channel by adding end point information for each interface such as complex Service References or simple URIs, digital certificates etc.

At run time, information about a channel variable is expanded further. This requires that the messages in the Choreography also contain Correlation information, for example by including:

- A SOAP header that specifies the correlation data to be used with the Channel, or
- Using the actual value of data within a message, for example the Order Number of the Order that is common to all the messages sent over the Channel

In practice, when a Choreography is performed, several different ways of doing correlation may be employed which vary depending on the *Channel Type*.

3.5.4 Interaction Operations

An Operation specifies the particular part of an interface that is the target for a message. The content varies depending on the type of choreography.

Choreography Type	Interaction
Abstract	In an abstract choreography, an operation is described by a unique name within the Interface within the Channel
Portable	In a portable choreography, an operation is described referencing a WSDL one-way or request-response Operation
Concrete	Same as portable.

3.5.5 Interaction State Changes

States contain information about the State of a Role as a result of information exchanged in the form of an *Interaction*. For example after an Interaction where an order is sent by a

Buyer to a Seller, the Buyer could create the *State Variable* “Order State“ and assign the value “Sent” when the message was sent, and when the Seller received the order, the Seller could also create its own version of the “Order State” *State Variable* and assign it the value “Received”.

As a result of a State Change, several different outcomes are possible which can only be determined at run time. The *Interaction* lists each of these allowed *State Changes*, for example when an order is sent from a Buyer to a Seller the outcomes could be one of the following *State Changes*:

1. Buyer.OrderState = Sent, Seller.OrderState = Received
2. Buyer.OrderState = SendFailure, Seller.OrderState not set
3. Buyer.OrderState = AckReceived, Seller.OrderState = OrderAckSent

3.5.6 Interaction Based Alignment

In some choreographies there may be a requirement that, at the end of an Interaction, the Roles in the Choreography have agreement of the outcome. More specifically within an Interaction, a Role needs to have a common understanding of the state changes of one or more *State Variables* that are complimentary to one or more *State Variables* of its partner Role. Additionally within an Interaction, a Role needs to have a common understanding of the values of the *Information Exchange Variables* at the partner Role.

Without alignment the Buyer knows that her “OrderState” is set to “Sent”, but does not know the value of the OrderState at the Seller. Once the Seller receives the Order then the Seller knows that his “OrderState” variable is set to “Received”. He also knows the Buyers “OrderState” was set to “Sent”, as the Choreography defines that the Buyer’s Order State variable is set in this way when an Order is sent.

With Choreography Alignment the difference is that both the Buyer and the Seller have:

- State Variables such as Order State variables at the Buyer and Seller, that have Values that are complementary to each other, e.g. Sent at the Buyer and Received at the Seller, and
- Knowledge of the values of each others States Variables, i.e. the Buyer knows that the Seller’s “OrderState” variable has the value “Received” and the Seller knows that the Buyer’s “OrderState” variable is set to “Sent”
- Information Exchange Variables that have the same types with the same content, e.g. The Order variables at the Buyer and Seller have the same Variable Types and hold the same order

This assurance of the outcome with respect to States is achieved by an ‘agreement’ protocol that is used in conjunction with the Choreography such as the Web Services and other specifications designed to coordinate long-running transactions.

3.5.7 Protocol Based Information Exchanges

The *One-Way*, *Request* or *Response* messages in an Interaction may also be implemented using a *Protocol Based Exchange* where a series of messages are exchanged according to some well-known protocol, such as the reliable messaging protocols defined in specifications such as WS Reliability.

In both cases, the same or similar *Message Content* may be exchanged as in a simple Interaction, for example the sending of an Order between a Buyer and a Seller. Therefore some of the same *State Changes* may result.

However when protocols such as the reliable messaging protocols are used, additional *State Changes* will occur. For example, if a reliable messaging protocol were being used then the Buyer, once confirmation of delivery of the message was received, would also know that the Seller's "Order State" variable was in the state “Received” even though there was no separate Interaction that described this.

Issue I-03. Do we add additional standard states to describe the outcomes of using reliable messaging protocols? Similarly, should we include additional states to handle other outcomes, such as security failures?

Issue I-04. Do we want to specify standard "reusable" names for the State Variable values for the common states associated with Interactions, e.g. for an Interaction you could have “Sent”, “Received”, “SendFailure”, et.c

3.6 Activities and Control Structures

The diagram below shows the model for Activities …

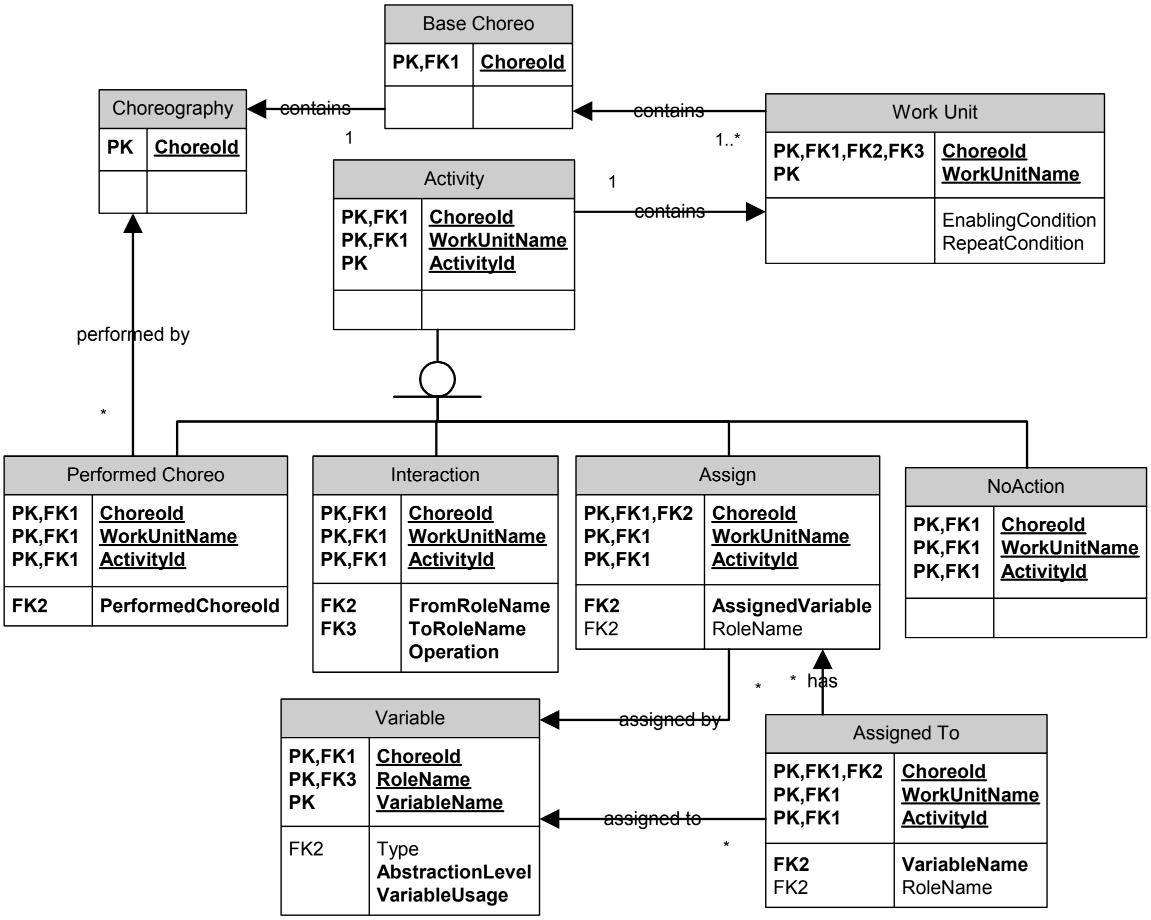

Figure 7: Model for Activities

… and this diagram shows the model for Control Structures …

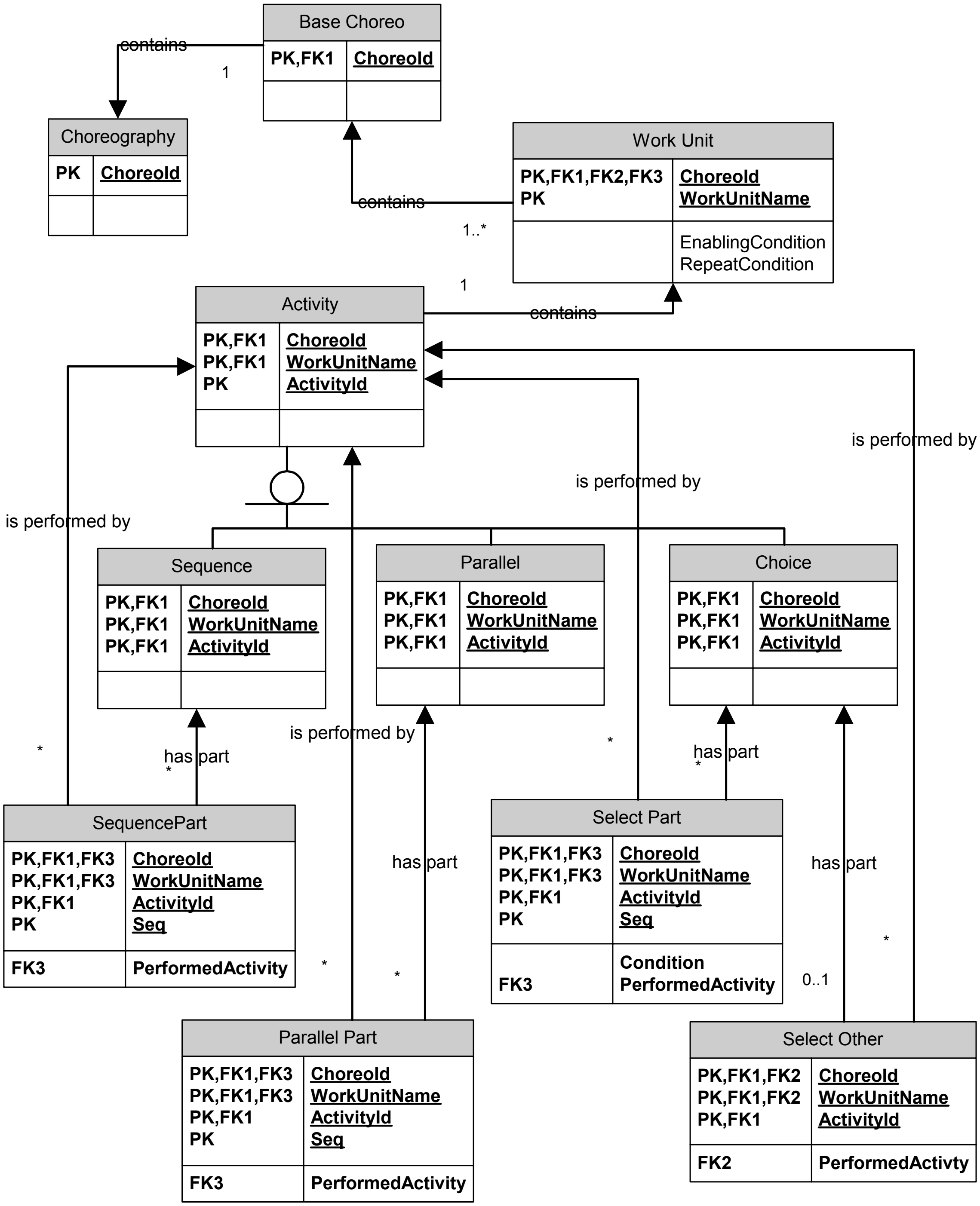

Figure 8: Model for Control Structures

Activities are the lowest level components of the Choreography which do the actual work, such as the Interactions described earlier or the performance of other Choreographies.

Control Structures combine these *Activities* with other *Control Structures* in a nested way to specify the sequence and flow of the exchange of information within the Choreography.

However at the highest level, the Choreographies consist of *Work Units*, that each contain a single Activity that is performed whenever an optional enabling condition on the *Work Unit*, called a *Guard*, is true.

Each *Activity* within a Work Unit is then either:

- A *Basic Activity* that does the actual work. These are:
    - An *Interaction*, i.e. the Work Unit consists of a single Interaction as described earlier
    - A *Perform,* which means that a complete, separately defined choreography is performed
    - An *Assign*, which assigns, within one Role, the value of one Variable to the value of a Variable or Token, or makes a Token reference a Variable or another Token
    - *No Action*, which means that the Choreography should take no particular action at that point
- *Control Structures* that group Basic Activities and Control Structures together in a nested structure to express the logic and decision flow involved in the Choreography. The Control Structures are:
    - *Sequence*, which specifies a list of Activities that are performed in sequence
    - *Choice*, which specifies that just one of two or more Activities are performed depending on the condition on a Guard
    - *Parallel*, which means that all the Activities are performed at the same time.

Note that an *Activity* is a modeling concept that would not appear in an XML equivalent of a Choreography definition. However it is needed to allow the Sequence, Choice and Parallel *Activities* to contain other *Activities* in a nested structure that allows the specification of *Work Units* that contain multiple *Activities* to be created.

Each of the ideas above, apart from Interaction, which was described earlier, is described in more detail below.

3.6.1 Work Units

Each Work Unit has two optional conditions or guards associated with it:

- An *Enabling Condition*, which specifies a combination of states that must be available and also evaluate to true before the Work Unit can be performed, and

- A *Repeat Condition* that specifies a combination of states that, if they evaluate to true once the Work Unit is complete, means that the enabling condition of the Work Unit (if specified) is re-checked.

Examples of a Work Unit include:

- A *Send PO* Work Unit that includes Interactions for the Buyer to send an Order, the Supplier to acknowledge the order, and then later accept (or reject) the order. This work unit would probably not have a guard
- An *Order Delivery Error* Work Unit that is performed whenever the *Place Order* Work Unit did not reach a “normal” conclusion. This would have a Guard condition that identifies the error – see also Choreography Exceptions and Transactions
- A *Change Order* Work Unit that can be performed whenever an order acknowledgement message has been received and an order rejection has not been received.

Issue ACS-01: How do you limit the number of repetitions of a work unit to a specific number where the number is fixed, or where the number of repetitions varies by instance or dependent on some “Locally Defined variable”.

3.6.2 Performed Choreography

The Performed Choreography Structure enables a Choreography to define that a separately defined Choreography is to be performed. The Choreography that is performed can be defined either within the same Choreography Definition or separately.

Issue ACS-02. Should variables be used to define:

- *How Message Content is passed to (or from) the performed Choreogaphy*
- *How state information is passed to (or from) a performed choreography*

3.6.3 Assign

Assign sets, within one Role, the value of one Variable to the value of a Variable or Token, or makes a Token reference a Variable or another Token. The assignments may include:

- Assigning one *Information Exchange Variable* to another, for example so that a Choreography can define that a message received by one role is forwarded to another
- Assigning a *Locally Defined Variable* to part of the data contained in an Information Exchange Variable

3.6.4 NoAction

This *Activity* means that the choreography at this point should take no particular action. Examples of its use include:

- In a *Work Unit*, when there is a need to wait until the Guard condition on the *Work Unit* is true, for example you need to wait until say three separate Interactions are complete before progressing to the next step in the Choreography
- In a *Choice* so that you can enumerate all the possible choices even if some of the choices involve no Interactions.

3.6.5 Sequence Control Structure

The Sequence Control Structure enables a *Work Unit* to define that one or more Activities must be performed in sequence. It works by grouping the *Activities* within a <Sequence> ... </Sequence>

Activities must be performed in the same sequence that they are defined.

3.6.6 Choice Control Structure

The Choice Control structure enables a *Work Unit* to define that only one of two or more Activities should be performed. It works by:

- Grouping the *Work Units* within a <Choice> ... </Choice>
- Adding a Guard statement to each individual Activity within the Choice.

An Activity should only occur if the Guard on the Activity evaluates to true. Once one of the *Activities* in the Choice have been performed, then no other *Activities* in the Choice must be performed.

3.6.7 Parallel Control Structure

The Parallel Control Structure enables a *Work Unit* to define that Activities are performed in parallel. It works by grouping the Activities within a <Parallel> ... </Parallel>

3.7 Choreography Exceptions and Transactions

The diagram below shows the model for Choreography Exceptions and Transactions.

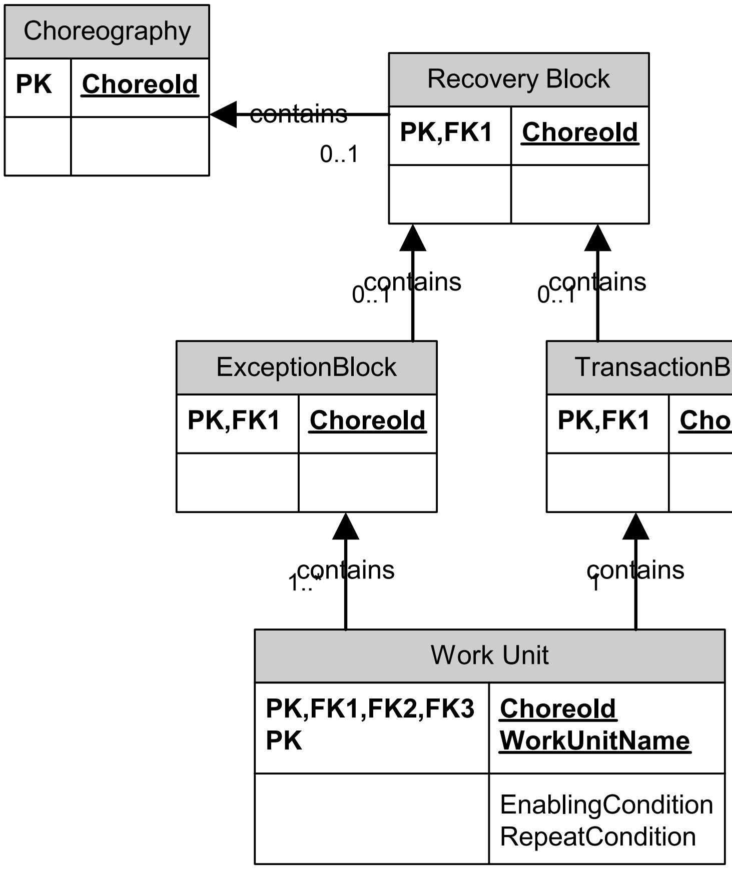

Figure 9: Model for Exceptions and Transactions

Choreographies are the unit for recovery purposes. This means that the Choreography should provide:

- *Exception Handling* so that Information Exchanges can be defined that are to be followed when an unexpected condition occurs while the Choreography is being performed
- *Transaction Handling* so that one Choreography can perform another Choreography up to a suitable point and then later perform the Choreography again to compensate for its effects.

To handle both of these a Choreography may contain an optional *Recovery Block* which contains one or both of:

- An *Exception Block*, that contains one or more *Work Units* with guards. If some exceptional circumstance occurs when the Choreography is performed one of the Work Units will be followed
- A *Transaction Block*, that contains a single *Work Unit* that is followed when the effects of the Choreography need to be reversed

Note however, that although recovery from errors can be defined, depending on the nature of the choreography and the problem that occurs, recovery, in practice may not be possible.

3.7.1 Exception Block

A Choreography can sometimes fail as a result of an exceptional circumstance or error. Different types of exceptions are possible including this non-exhaustive list:

- *Interaction Failures*, for example the sending of a message did not occur
- *Protocol Based Exchange failures*, for example no acknowledgement was received as part of a reliable messaging protocol
- *Security failures*, for example a Message was rejected by a recipient because the digital signature was not valid
- *Choreography Sequence Failures*, for example a Message was received that was not in the sequence as defined by the Choreography
- *Timeout errors*, for example an Interaction did not complete within a required timescale
- *Validation Errors*, for example an XML order document was not well formed or did not conform to its schema definition
- *Business Process "failures"*, for example the goods ordered were out of stock.

To handle these and other "errors" separate Work Units are defined in the *Exception Block* for each "exception" condition (as identified by its guards) that needs to be handled. Only one Work Unit per exception should be performed.

Issue CET-01. What happens if you get an error in a Work Unit that is within an Exception Block

Issue CET-02. What happens if you get an error condition for which no Work Unit in an Exception Block is specified

Issue CET-03. Should you be able to resume a choreography where the exception occurred if the exception block managed, for example, to fix the problem

Issue CET-04. How do you indicate that the choreography completed with an error if the choreography is being performed

Issue CET-05. Do we need "special" standard conditions/states that correspond to such things as "choreography out of sequence" that are choreography language specific

Issue CET-06. What does exception matching in the Work Unit guard condition?

Issue CET-07. Do we want to allow "catch all" conditions in Work Units in an Exception Block so that you can define the behavior of the Choreography when common but unlikely errors occur? For example, you could have an enabling condition that looked something like ".StateVariable=SendFailure".*

3.7.2 Transaction Block

Transaction handling allows one Choreography to perform another Choreography up to a suitable point and then later perform the Choreography again to compensate for its effects.

For example, a Choreography could exist that supported the purchasing of a property that involved exchanging information between many Roles including: a Purchaser who was buying the property, a Seller who was selling the property, a Buyers Agent who was advising the Purchaser, a Bank that was going to provide a mortgage, and a Title Company to prepare all the legal documents and handle the transfer of the actual moneys.

Only when the purchase was completed, could the choreography fully complete, and if the purchase fell through, then the effects of the Choreography would need to be reversed at each Role.

The Choreography model supports this type of example by defining a *Transaction Block* that contains one Work Unit that is followed when the effects of the Choreography need to be reversed.

In the example given above, this would work by defining a Choreography that:

- Performs one Choreography to carry out the main part of the process of creating and exchanging information required for the transaction
- Waits until the critical messages have been received that indicate that the transaction is either going to go ahead or fail
- Performs the main choreography again indicating that the effect of the Choreography should be Compensated by performing a *Work Unit* in the *Transaction Block*.

3.8 Semantics

Although not shown on this model, descriptions will be required to allow the recording of semantics definitions. In principle, this will be supported by including a *Description* structure in the definition of almost every single component within the model.

This *Description* structure will allow for the recording of semantics in any or all of the following ways:

- *Text.* This will be in plain text or possibly HTML and should be brief.
- *Document Reference*. This will contain a URL to a document that more fully describes the component. For example on the top level Choreography Definition that might reference a complete paper
- *Structured Attributes.* This will contain a machine processable definition in a languages such as RDF or OWL.

Descriptions that are *Text* or *Document References* can be defined in multiple different human readable languages.