Web Services Choreography Description Language, Version 1.0

Editor's Draft, 08 December 2004

This version:
TBD
Latest version:
TBD
Previous Version:
Not Applicable
Editors:
Nickolas Kavantzas, Oracle
David Burdett, Commerce One
Gregory Ritzinger, Novell
Yves Lafon, W3C

Copyright © 2004 W3C® (MIT, ERCIM, Keio), All Rights Reserved. W3C liability, trademark, document use and software licensing rules apply.

Abstract

The Web Services Choreography Description Language (WS-CDL) is an XML-based language that describes peer-to-peer collaborations of parties by defining, from a global viewpoint, their common and complementary observable behavior; where ordered message exchanges result in accomplishing a common business goal.

The Web Services specifications offer a communication bridge between the heterogeneous computational environments used to develop and host applications. The future of E-Business applications requires the ability to perform long-lived, peer-to-peer collaborations between the participating services, within or across the trusted domains of an organization.

The Web Services Choreography specification is targeted for composing interoperable, peer-to-peer collaborations between any type of party regardless

32 of the supporting platform or programming model used by the implementation of
33 the hosting environment.

34 Status of this Document

35 This section describes the status of this document at the time of its publication.
36 Other documents may supersede this document. A list of current W3C
37 publications and the latest revision of this technical report can be found in the
38 W3C technical reports index at http://www.w3.org/TR/.

39 This is the 3rd Public Working Draft of the Web Services Choreography
40 Description Language document.

41 It has been produced by the Web Services Choreography Working Group, which
42 is part of the Web Services Activity. This document represents consensus within
43 the Working Group about the Web Services Choreography description language.

44 This document is a chartered deliverable of the Web Services Choreography
45 Working Group.

46 Comments on this document should be sent to public-ws-chor-
47 comments@w3.org (public archive). It is inappropriate to send discussion emails
48 to this address.

49 Discussion of this document takes place on the public public-ws-chor@w3.org
50 mailing list (public archive) per the email communication rules in the Web
51 Services Choreography Working Group charter.

52 This document has been produced under the 24 January 2002 CPP as amended
53 by the W3C Patent Policy Transition Procedure. An individual who has actual
54 knowledge of a patent which the individual believes contains Essential Claim(s)
55 with respect to this specification should disclose the information in accordance
56 with section 6 of the W3C Patent Policy. Patent disclosures relevant to this
57 specification may be found on the Working Group's patent disclosure page.

58 Publication as a Working Draft does not imply endorsement by the W3C
59 Membership. This is a draft document and may be updated, replaced or
60 obsoleted by other documents at any time. It is inappropriate to cite this
61 document as other than work in progress.

62 Revision Description

63 This is the 4th editor's draft of the Web Services Choreography Description
64 Language document.

65 # Table of Contents

66 Status of this Document 2
67 Revision Description 2
68 1 Introduction 5
69 1.1 Notational Conventions 6
70 1.2 Purpose of the Choreography Description Language 8
71 1.3 Goals 9
72 1.4 Relationship with XML and WSDL 11
73 1.5 Relationship with Business Process Languages 11
74 1.6 Time Assumptions 11
75 2 Choreography Description Language Model 11
76 2.1 WS-CDL Model Overview 12
77 2.2 WS-CDL Document Structure 13
78 2.2.1 Choreography Package 13
79 2.2.2 Including WS-CDL Type Definitions 14
80 2.2.3 WS-CDL document Naming and Linking 15
81 2.2.4 Language Extensibility and Binding 15
82 2.2.5 Semantics 15
83 2.3 Collaborating Parties 16
84 2.3.1 Role Types 16
85 2.3.2 Relationship Types 17
86 2.3.3 Participant Types 18
87 2.3.4 Channel Types 19
88 2.4 Information Driven Collaborations 21
89 2.4.1 Information Types 21
90 2.4.2 Variables 23
91 2.4.3 Expressions 25
92 2.4.3.1 WS-CDL Supplied Functions 25
93 2.4.4 Tokens 27
94 2.4.5 Choreographies 29
95 2.4.6 WorkUnits 31
96 2.4.7 Choreography Life-line 35
97 2.4.8 Choreography Exception Handling 36
98 2.4.9 Choreography Finalization 38
99 2.4.10 Choreography Coordination 39
100 2.5 Activities 42
101 2.5.1 Ordering Structures 43
102 2.5.1.1 Sequence 43
103 2.5.1.2 Parallel 43
104 2.5.1.3 Choice 43
105 2.5.2 Interacting 44
106 2.5.2.1 Interaction Based Information Alignment 45
107 2.5.2.2 Interaction Life-line 46
108 2.5.2.3 Interaction Syntax 46
109 2.5.3 Composing Choreographies 53

110 2.5.4 Assigning Variables 56
111 2.5.5 Marking Silent Actions 57
112 2.5.6 Marking the Absence of Actions 58
113 2.5.7 Finalizing a Choreography 58
114 3 Example 60
115 4 Relationship with the Security framework 61
116 5 Relationship with the Reliable Messaging framework 61
117 6 Relationship with the Coordination framework 61
118 7 Relationship with the Addressing framework 61
119 8 Conformance 62
120 9 Acknowledgments 62
121 10 References 63
122 11 Last Call Issues 65
123 11.1 Issue 1 65
124 11.2 Issue 2 65
125 12 WS-CDL XSD Schemas 66
126

126 # 1 Introduction

127 For many years, organizations have being developing solutions for automating
128 their peer-to-peer collaborations, within or across their trusted domain, in an
129 effort to improve productivity and reduce operating costs.

130 The past few years have seen the Extensible Markup Language (XML) and the
131 Web Services framework developing as the de facto choices for describing
132 interoperable data and platform neutral business interfaces, enabling more open
133 business transactions to be developed.

134 Web Services are a key component of the emerging, loosely coupled, Web-
135 based computing architecture. A Web Service is an autonomous, standards-
136 based component whose public interfaces are defined and described using XML.
137 Other systems may interact with a Web Service in a manner prescribed by its
138 definition, using XML based messages conveyed by Internet protocols.

139 The Web Services specifications offer a communication bridge between the
140 heterogeneous computational environments used to develop and host
141 applications. The future of E-Business applications requires the ability to perform
142 long-lived, peer-to-peer collaborations between the participating services, within
143 or across the trusted domains of an organization.

144 The Web Service architecture stack targeted for integrating interacting
145 applications consists of the following components:

146 • *SOAP*: defines the basic formatting of a message and the basic delivery
147 options independent of programming language, operating system, or
148 platform. A SOAP compliant Web Service knows how to send and receive
149 SOAP-based messages

150 • *WSDL*: describes the static interface of a Web Service. It defines the
151 message set and the message characteristics of end points. Data types
152 are defined by XML Schema specification, which supports rich type
153 definitions and allows expressing any kind of XML type requirement for the
154 application data

155 • *Registry*: allows publishing the availability of a Web Service and its
156 discovery from service requesters using sophisticated searching
157 mechanims

158 • *Security layer:* ensures that exchanged information are not modified or
159 forged in a verifiable manner and that parties can be authenticated

160 • *Reliable Messaging layer:* provides exactly-once and guaranteed delivery
161 of information exchanged between parties

162 • *Context, Coordination and Transaction layer*: defines interoperable
163 mechanisms for propagating context of long-lived business transactions

164 and enables parties to meet correctness requirements by following a
165 global agreement protocol

166 • *Business Process Languages layer*: describes the execution logic of Web
167 Services based applications by defining their control flows (such as
168 conditional, sequential, parallel and exceptional execution) and prescribing
169 the rules for consistently managing their non-observable data

170 • *Choreography layer*: describes collaborations of parties by defining from a
171 global viewpoint their common and complementary observable behavior,
172 where information exchanges occur, when the jointly agreed ordering
173 rules are satisfied

174 The Web Services Choreography specification is aimed at the composition of
175 interoperable collaborations between any type of party regardless of the
176 supporting platform or programming model used by the implementation of the
177 hosting environment.

178 1.1 Notational Conventions

179 The keywords "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT",
180 "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in
181 this document are to be interpreted as described in RFC-2119 [2].

182 The following namespace prefixes are used throughout this document:

Prefix	Namespace URI	Definition
wsdl	http://www.w3.org/2004/08/wsdl	WSDL 2.0 namespace for WSDL framework.
cdl	http://www.w3.org/2004/12/ws-chor/cdl	WSCDL namespace for Choreography Description Language.
xsi	http://www.w3.org/2001/XMLSchema-instance	Instance namespace as defined by XSD [11].
xsd	http://www.w3.org/2001/XMLSchema	Schema namespace as defined by XSD [12].

tns	(various)	The "this namespace" (tns) prefix is used as a convention to refer to the current document.
(other)	(various)	All other namespace prefixes are samples only. In particular, URIs starting with "http://sample.com" represent some application-dependent or context-dependent URIs [4].

183 This specification uses an *informal syntax* to describe the XML grammar of a
184 WS-CDL document:

185 • The syntax appears as an XML instance, but the values indicate the data
186 types instead of values.

187 • Characters are appended to elements and attributes as follows: "?" (0 or
188 1), "*" (0 or more), "+" (1 or more).

189 • Elements names ending in "…" (such as <element…/> or <element…>)
190 indicate that elements/attributes irrelevant to the context are being
191 omitted.

192 • Grammar in bold has not been introduced earlier in the document, or is of
193 particular interest in an example.

194 • <-- extensibility element --> is a placeholder for elements from some
195 "other" namespace (like ##other in XSD).

196 • The XML namespace prefixes (defined above) are used to indicate the
197 namespace of the element being defined.

198 • Examples starting with <?xml contain enough information to conform to
199 this specification; other examples are fragments and require additional
200 information to be specified in order to conform.

201 An XSD is provided as a formal definition of WS-CDL grammar (see Section 11).

1.2 Purpose of the Choreography Description Language

Business or other activities that involve different organizations or independent processes are engaged in a collaborative fashion to achieve a common business goal, such as *Order Fulfillment*.

For the collaboration to work successfully, the rules of engagement between all the interacting parties must be provided. Whereas today these rules are frequently written in English, a standardized way for precisely defining these interactions, leaving unambiguous documentation of the parties and responsibilities of each, is missing.

The Web Services Choreography specification is aimed at being able to precisely describe collaborations between any type of party regardless of the supporting platform or programming model used by the implementation of the hosting environment.

Using the Web Services Choreography specification, a contract containing a "global" definition of the common ordering conditions and constraints under which messages are exchanged, is produced that describes, from a global viewpoint, the common and complementary observable behavior of all the parties involved. Each party can then use the global definition to build and test solutions that conform to it. The global specification is in turn realized by combination of the resulting local systems, on the basis of appropriate infrastructure support.

The advantage of a contract based on a global viewpoint as opposed to anyone endpoint is that it separates the overall "global" process being followed by an individual business or system within a "domain of control" (an endpoint) from the definition of the sequences in which each business or system exchanges information with others. This means that, as long as the "observable" sequences do not change, the rules and logic followed within a domain of control (endpoint) can change at will and interoperability is therefore guaranteed.

In real-world scenarios, corporate entities are often unwilling to delegate control of their business processes to their integration partners. Choreography offers a means by which the rules of participation within a collaboration can be clearly defined and agreed to, jointly. Each entity may then implement its portion of the Choreography as determined by the common or global view. It is the intent of CDL that the conformance of each implementation to the common view expressed in CDL is easy to determine.

The figure below demonstrates a possible usage of the Choreography Description Language.

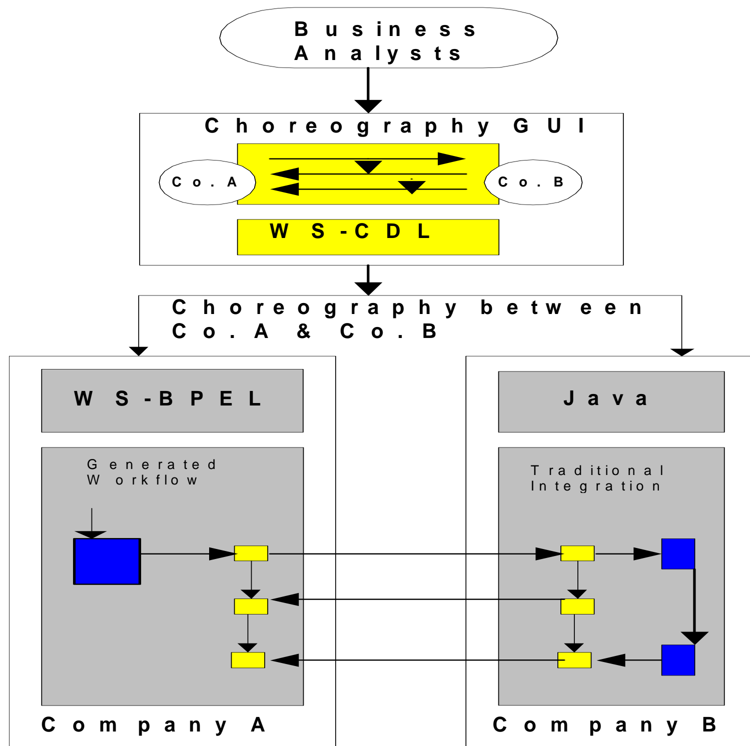

Figure 1: Integrating Web Services based applications using WS-CDL

In Figure 1, Company A and Company B wish to integrate their Web Services based applications. The respective business analysts at both companies agree upon the services involved in the collaboration, their interactions, and their common ordering and constraint rules under which the interactions occur. They then generate a Choreography Description Language based representation. In this example, a Choreography specifies the interactions between services across business entities ensuring interoperability, while leaving actual implementation decisions in the hands of each individual company:

- Company "A" relies on a WS-BPEL [18] solution to implement its own part of the Choreography
- Company "B", having greater legacy driven integration needs, relies on a J2EE [25] solution incorporating Java and Enterprise Java Bean Components or a .NET [26] solution incorporating C# to implement its own part of the Choreography

Similarly, a Choreography can specify the interoperability and interactions between services within one business entity.

1.3 Goals

The primary goal of a Choreography Description Language is to specify a declarative, XML based language that defines from a global viewpoint the common and complementary observable behavior specifically, the information

exchanges that occur and the jointly agreed ordering rules that need to be satisfied.

More specifically, the goals of the Choreography Description Language are to permit:

- *Reusability*. The same Choreography definition is usable by different parties operating in different contexts (industry, locale, etc.) with different software (e.g. application software)
- *Cooperation*. Choreographies define the sequence of exchanging messages between two (or more) independent parties or processes by describing how they should cooperate
- *Multi-Party Collaboration*. Choreographies can be defined involving any number of parties 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
- *Modularity*. Choreographies can be defined using an "inclusion" facility that allows a Choreography to be created from parts contained in several different Choreographies
- *Information Driven Collaboration*. Choreographies describe how parties make progress within a collaboration, through the recording of exchanged information and changes to observable information that cause ordering constraints to be fulfilled and progress to be made
- *Information Alignment*. Choreographies allow the parties that take part in Choreographies to communicate and synchronize their observable information
- *Exception Handling*. Choreographies can define how exceptional or unusual conditions that occur while the Choreography is performed are handled
- *Transactionality*. The processes or parties that take part in a Choreography can work in a "transactional" way with the ability to coordinate the outcome of the long-lived collaborations, which include multiple participants, each with their own, non-observable business rules and goals
- *Specification Composability*. This specification will work alongside and complement other specifications such as the WS-Reliability [22], WS-Composite Application Framework (WS-CAF) [21], WS-Security [24], Business Process Execution Language for WS (WS-BPEL) [18], etc.

299 ## 1.4 Relationship with XML and WSDL

300 The WS-CDL specification depends on the following specifications: XML 1.0 [9],
301 XML-Namespaces [10], XML-Schema 1.0 [11, 12] and XPath 1.0 [13]. Support
302 for including and referencing service definitions given in WSDL 2.0 [7] is a
303 normative part of the WS-CDL specification. In addition, support for including and
304 referencing service definitions given in WSDL 1.1 as constrained by WS-I Basic
305 Profile [Action: add references] is a normative part of the WS-CDL specification.

306 ## 1.5 Relationship with Business Process Languages

307 A Choreography Description Language is not an "executable business process
308 description language" or an implementation language. The role of specifying the
309 execution logic of an application will be covered by these specifications [16, 17,
310 18, 19, 20, 23, 26].

311 A Choreography Description Language does not depend on a specific business
312 process implementation language. Thus, it can be used to specify truly
313 interoperable, collaborations between any type of party regardless of the
314 supporting platform or programming model used by the implementation of the
315 hosting environment. Each party, adhering to a Choreography Description
316 Language collaboration representation, could be implemented using completely
317 different mechanisms such as:

318 - Applications, whose implementation is based on executable business
319 process languages [16, 17, 18, 19, 20]
320 - Applications, whose implementation is based on general purpose
321 programming languages [23, 26]
322 - Or human controlled software agents

323 ## 1.6 Time Assumptions

324 Clock synchronization is unspecified in the WS-CDL technical specification and is
325 considered design-specific. In specific environments between involved parties, it
326 can be assumed that all parties are reasonably well synchronized on second time
327 boundaries. However, finer grained time synchronization within or across parties,
328 or additional support or control are undefined and outside the scope of the WS-
329 CDL specification.

330 # 2 Choreography Description Language Model

331 This section introduces the Web Services Choreography Description Language
332 (WS-CDL) model.

2.1 WS-CDL Model Overview

WS-CDL describes interoperable, collaborations between parties. In order to facilitate these collaborations, services commit to mutual responsibilities by establishing Relationships. Their collaboration takes place in a jointly agreed set of ordering and constraint rules, whereby information is exchanged between the parties.

The WS-CDL model consists of the following entities:

- *Participant Types, Role Types and Relationship Types* - Within a Choreography, information is always exchanged between parties within or across trust boundaries. A Role Type enumerates the observable behavior a party exhibits in order to collaborate with other parties. A Relationship Type identifies the mutual commitments that must be made between two parties for them to collaborate successfully. A Participant Type is grouping together those parts of the observable behavior that must be implemented by the same logical entity or organization
- *Information Types, Variables and Tokens* - Variables contain information about commonly observable objects in a collaboration, such as the information exchanged or the observable information 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 contains or the Token references
- *Choreographies* - Choreographies define collaborations between interacting parties:
    - *Choreography Life-line* – The Choreography Life-line expresses the progression of a collaboration. Initially, the collaboration is established between parties, then work is performed within it and finally it completes either normally or abnormally
    - *Choreography Exception Blocks* - An Exception Block specifies what additional interactions should occur when a Choreography behaves in an abnormal way
    - *Choreography Finalizer Blocks* - A Finalizer Block describes how to specify additional interactions that should occur to modify the effect of an earlier successfully completed Choreography, for example to confirm or undo the effect
- *Channels* - A Channel realizes a point of collaboration between parties by specifying where and how information is exchanged
- *Work Units* - A Work Unit prescribes the constraints that must be fulfilled for making progress and thus performing actual work within a Choreography
- *Activities and Ordering Structures* - Activities are the lowest level components of the Choreography that perform the actual work. Ordering Structures combine activities with other Ordering Structures in a nested

structure to express the ordering conditions in which information within the Choreography is exchanged

- *Interaction Activity* - An Interaction is the basic building block of a Choreography, which results in an exchange of information between parties and possible synchronization of their observable information changes and the actual values of the exchanged information
- *Semantics* - Semantics allow the creation of descriptions that can record the semantic definitions of every component in the model

2.2 WS-CDL Document Structure

A WS-CDL document is simply a set of definitions. Each definition is a named construct that can be referenced. There is a *package* element at the root, and the individual Choreography type definitions inside.

2.2.1 Choreography Package

A *Choreography Package* aggregates a set of WS-CDL type definitions, provides a namespace for the definitions and through the use of XInclude [27], MAY syntactically include WS-CDL type definitions that are defined in other Choreography Packages.

The syntax of the *package* construct is:

```
<package
   name="ncname"
   author="xsd:string"?
   version="xsd:string"?
   targetNamespace="uri"
   xmlns=”http://www.w3.org/2004/12/ws-chor/cdl”>

   informationType*
   token*
   tokenLocator*
   roleType*
   relationshipType*
   participantType*
   channelType*

   Choreography-Notation*
</package>
```

The Choreography Package contains:

- Zero or more Information Types
- Zero or more Tokens and Token Locators

- Zero or more Role Types
- Zero or more Relationship Types
- Zero or more Participant Types
- Zero or more Channel Types
- Zero or more Package-level Choreographies

The top-level attributes name, author, and version define authoring properties of the Choreography document.

The targetNamespace attribute provides the namespace associated with all WS-CDL type definitions contained in this Choreography Package. WS-CDL type definitions included in this Package, using the inclusion mechanism, MAY be associated with other namespaces.

The elements informationType, token, tokenLocator, roleType, relationshipType, participantType and channelType MAY be used as elements by all the Choreographies defined within this Choreography Package.

2.2.2 Including WS-CDL Type Definitions

WS-CDL type definitions or fragments of WS-CDL type definitions can be syntactically reused in any WS-CDL type definition by using XInclude [27]. The assembly of large WS-CDL type definitions from multiple smaller, well-formed WS-CDL type definitions or WS-CDL type definitions fragments is enabled using this mechanism.

Inclusion of fragments of other WS-CDL type definitions SHOULD be done carefully in order to avoid duplicate definitions (Variables, blocks, etc.). A WS-CDL processor MUST ensure that the document is correct before processing it. The correctness may involve XML well-formedness as well as semantic checks, such as unicity of Variable definitions, of a single root Choreography, etc.

The example below shows some possible syntactic reuses of Choreography type definitions.

```
<choreography name="newChoreography” root="true">
...
   <variable name="newVariable" informationType="someType"
             role="randomRome"/>
   <xi:include href="genericVariableDefinitions.xml" />
   <xi:include href="otherChoreography.xml"
               xpointer="xpointer(//choreography/variable[1]) />
...
</choreography>
```

454 ### 2.2.3 WS-CDL document Naming and Linking

455 WS-CDL documents MUST be assigned a name attribute of type NCNAME that
456 serves as a lightweight form of documentation.

457 The targetNamespace attribute of type URI MUST be specified.

458 The URI MUST NOT be a relative URI.

459 A reference to a definition is made using a QName.

460 Each definition type has its own name scope.

461 Names within a name scope MUST be unique within a WS-CDL document.

462 The resolution of QNames in WS-CDL is similar to the resolution of QNames
463 described by the XML Schemas specification [11].

464 ### 2.2.4 Language Extensibility and Binding

465 To support extending the WS-CDL language, this specification allows the use of
466 extensibility elements and/or attributes defined in other XML namespaces.
467 Extensibility elements and/or attributes MUST use an XML namespace different
468 from that of WS-CDL. All extension namespaces used in a WS-CDL document
469 MUST be declared.

470 Extensions MUST NOT change the semantics of any element or attribute from
471 the WS-CDL namespace.

472 ### 2.2.5 Semantics

473 Within a WS-CDL document, descriptions allow the recording of semantic
474 definitions and other documentation. The OPTIONAL description sub-element is
475 allowed inside any WS-CDL language element. WS-CDL parsers are not
476 required to parse the contents of the description.

477 The information provided by the description sub-element will allow for the recording
478 of semantics in any or all of the following ways:

479 - *Text.* This will be in plain text or possibly HTML and should be brief
480 - *Document Reference*. This will contain a URI to a document that more
481 fully describes the component
482 - *Machine Oriented Semantic Descriptions.* This will contain machine
483 processable definitions in languages such as RDF or OWL

484 Descriptions that are text or document references can be defined in multiple
485 different human readable languages.

2.3 Collaborating Parties

The WSDL specification [7] describes the functionality of a service provided by a party based on a stateless, client-server model. The emerging Web Based applications require the ability to exchange information in a peer-to-peer environment. In these types of environments a party represents a requester of services provided by another party and is at the same time a provider of services requested from other parties, thus creating mutual multi-party service dependencies.

A WS-CDL document describes how a party is capable of engaging in collaborations with the same party or with different parties.

The *Role Types*, *Participant Types*, *Relationship Types* and *Channel Types* define the coupling of the collaborating parties.

2.3.1 Role Types

A *Role Type* enumerates the observable behavior a party exhibits in order to collaborate with other parties. For example the “Buyer” Role Type is associated with purchasing of goods or services and the “Supplier” Role Type is associated with providing those goods or services for a fee.

The syntax of the *roleType* construct is:

```
<roleType name="ncname">
   <behavior name="ncname" interface="qname"? />+
</roleType>
```

The attribute name is used for specifying a distinct name for each roleType element declared within a Choreography Package.

Within the roleType element, the behavior element specifies a subset of the observable behavior a party exhibits. A Role Type MUST contain one or more behavior elements. The attribute name within the behavior element is used for specifying a distinct name for each behavior element declared within a roleType element.

The behavior element defines an OPTIONAL interface attribute, which identifies a WSDL interface type. A behavior without an interface describes a Role Type that is not required to support a specific Web Service interface.

520 ### 2.3.2 Relationship Types

521 A *Relationship Type* identifies the Role Types and Behaviors, where mutual
522 commitments between two parties MUST be made for them to collaborate
523 successfully. For example the Relationship Types between a Buyer and a Seller
524 could include:

525 - A "Purchasing" Relationship Type, for the initial procurement of goods or
526 services, and
527 - A "Customer Management" Relationship Type to allow the Supplier to
528 provide service and support after the goods have been purchased or the
529 service provided

530 Although Relationship Types are always between two Role Types,
531 Choreographies involving more than two Role Types are possible. For example if
532 the purchase of goods involved a third-party Shipper contracted by the Supplier
533 to deliver the Supplier's goods, then, in addition to the "Purchasing" and
534 "Customer Management" Relationship Types described above, the following
535 Relationship Types might exist:

536 - A "Logistics Provider" Relationship Type between the Supplier and the
537 Shipper, and
538 - A "Goods Delivery" Relationship Type between the Buyer and the Shipper

539

540 The syntax of the *relationshipType* construct is:
541

```
542 <relationshipType name="ncname">
543    <role type="qname" behavior="list of ncname"? />
544    <role type="qname" behavior="list of ncname"? />
545 </relationshipType>
```

546

547 The attribute name is used for specifying a distinct name for each relationshipType
548 element declared within a Choreography Package.

549 A relationshipType element MUST have exactly two Role Types defined. Each Role
550 Type is specified by the type attribute within the role element.

551 Within each role element, the OPTIONAL attribute behavior identifies the
552 commitment of a party as an XML-Schema list of behavior types belonging to this
553 Role Type. If the behavior attribute is missing then all the behaviors belonging to
554 this Role Type are identified as the commitment of a party.

2.3.3 Participant Types

A *Participant Type* identifies a set of Role Types that MUST be implemented by the same logical entity or organization. Its purpose is to group together the parts of the observable behavior that MUST be implemented by the same logical entity or organization.

The syntax of the *participantType* construct is:

```
<participantType name="ncname">
   <role type="qname" />+
</participantType>
```

The attribute name is used for specifying a distinct name for each participantType element declared within a Choreography Package.

Within the participantType element, one or more role elements identify the Role Types that MUST be implemented by this Participant Type. Each Role Type is specified by the type attribute of the role element. A specific Role Type MUST NOT be specified in more than one participantType element.

An example is given below where the “SellerForBuyer” Role Type belonging to a “Buyer-Seller” Relationship Type is implemented by the Participant Type “Broker” which also implements the “SellerForShipper” Role Type belonging to a “Seller-Shipper” Relationship Type:

```
<roleType name="Buyer">
   …
</roleType>
<roleType name="SellerForBuyer">
    <behavior name="sellerForBuyer" interface="rns:sellerForBuyerPT"/>
</roleType>
<roleType name="SellerForShipper">
    <behavior name="sellerForShipper" interface="rns:sellerForShipperPT"/>
</roleType>
<roleType name="Shipper">
   …
</roleType>
<relationshipType name="Buyer-Seller">
    <role type="tns:Buyer" />
    <role type="tns:SellerForBuyer" />
</relationshipType>
<relationshipType name="Seller-Shipper">
    <role type="tns:SellerForShipper" />
    <role type="tns:Shipper" />
</relationshipType>

<participantType name="Broker">
   <role type="tns:SellerForBuyer" />
```

602 603

```
</participantType>
```

604

2.3.4 Channel Types

605 A *Channel* realizes a point of collaboration between parties by specifying where
606 and how information is exchanged between collaborating parties. Additionally,
607 Channel information can be passed among parties in information exchanges.
608 The Channels exchanged MAY be used in subsequent Interaction activities. This
609 allows the modeling of both static and dynamic message destinations when
610 collaborating within a Choreography. For example, a Buyer could specify
611 Channel information to be used for sending delivery information. The Buyer could
612 then send the Channel information to the Seller who then forwards it to the
613 Shipper. The Shipper could then send delivery information directly to the Buyer
614 using the Channel information originally supplied by the Buyer.

615 A Channel Type MUST describe the Role Type and the reference type of a party,
616 being the target of an information exchange, which is then used for determining
617 where and how to send or receive information to or into the party.

618 A Channel Type MAY specify the instance identity of an entity implementing the
619 behavior(s) of a party, being the target of an information exchange.

620 A Channel Type MAY describe one or more logical conversations between
621 parties, where each conversation groups a set of related information exchanges.

622 One or more Channel(s) MAY be passed around from one party to another in an
623 information exchange. A Channel Type MAY be used to:

624 - Restrict the number of times a Channel of this Channel Type can be used
625 - Restrict the type of information exchange that can be performed when
626 using a Channel of this Channel Type
627 - Restrict the Channel Type(s) that will be passed through a Channel of this
628 Channel Type
629 - Enforce that a passed Channel is always distinct

630

631 The syntax of the *channelType* construct is:
632

633–643

```
<channelType  name="ncname"
              usage="once"|"unlimited"?
              action="request-respond"|"request"|"respond"? >

   <passing  channel="qname"
             action="request-respond"|"request"|"respond"?
             new="true"|"false"? />*

   <role  type="qname"  behavior="ncname"? />

   <reference>
```

```
   </reference>

   <identity>
      <token name="qname"/>+
   </identity>?
</channelType>
```

The attribute name is used for specifying a distinct name for each channelType element declared within a Choreography Package.

The OPTIONAL attribute usage is used to restrict the number of times a Channel of this Channel Type can be used.

The OPTIONAL attribute action is used to restrict the type of information exchange that can be performed when using a Channel of this Channel Type. The type of information exchange performed could either be a request-respond exchange, a request exchange, or a respond exchange. The default for this attribute is set to “request”.

The OPTIONAL element passing describes the Channel Type(s) of the Channel(s) that are passed, from one party to another, when using an information exchange on a Channel of this Channel Type. The OPTIONAL attribute action within the passing element defines if a Channel will be passed during a request exchange, during a response exchange or both. The default for this attribute is set to “request”. The OPTIONAL attribute new within the passing element when set to “true” enforces a passed Channel to be always distinct. If the element passing is missing then this Channel Type MAY be used for exchanging information but MUST NOT be used for passing Channels of any Channel Type.

The element role is used to identify the Role Type of a party, being the target of an information exchange, which is then used for statically determining where and how to send or receive information to or into the party.

The element reference is used for describing the reference type of a party, being the target of an information exchange, which is then used for dynamically determining where and how to send or receive information to or into the party. The reference of a party is distinguished by a Token as specified by the name attribute of the token element within the reference element.

The OPTIONAL element identity MAY be used for identifying an instance of an entity implementing the behavior of a party and for identifying a logical conversation between parties. The identity and the different conversations are distinguished by a set of Tokens as specified by the name attribute of the token element within the identity element.

The following rule applies for Channel Type:

- If two or more Channel Types SHOULD point to Role Types that MUST be implemented by the same logical entity or organization, then the specified Role Types MUST belong to the same Participant Type. In addition, the

identity elements within the Channel Types MUST have the same number of Tokens with the same informationTypes specified in the same order

The example below shows the definition of the Channel Type “RetailerChannel” that realizes a point of collaboration with a Retailer. The Channel Type identifies the Role Type of the Retailer as the “Retailer”. The information for locating the Retailer is specified in the reference element, whereas the instance of a process implementing the Retailer is identified for correlation purposes using the identity element. The element passing allows only a Channel of “ConsumerChannel” Type to be passed in a request information exchange through a Channel of “RetailerChannel” Type.

```
<channelType name="RetailerChannel">
   <passing channel="ConsumerChannel" action="request" />

   <role type="tns:Retailer" behavior="retailerForConsumer"/>

   <reference>
     <token name="tns:retailerRef"/>
   </reference>

   <identity>
     <token name="tns:purchaseOrderID"/>
   </identity>
</channelType>
```

2.4 Information Driven Collaborations

Parties make progress within a collaboration when recordings of exchanged information are made, and changes to observable information occur, that then cause ordering constraints to be fulfilled. A WS-CDL document allows defining information within a Choreography that can influence the observable behavior of the collaborating parties.

Variables capture information about objects in the Choreography, such as the information exchanged or the observable information of the Roles involved. *Tokens* are aliases that can be used to reference parts of a Variable. Both Variables and Tokens have *Information Types* that define the type of information the Variable contains or the Token references.

2.4.1 Information Types

Information Types describe the type of information used within a Choreography. By introducing this abstraction, a Choreography definition avoids referencing directly the data types, as defined within a WSDL document or an XML Schema document.

730 The syntax of the *informationType* construct is:
731

```
732 <informationType name="ncname"
733                  type="qname"?|element="qname"?
734                  exceptionType="true"|"false"? />
```

735

736 The attribute name is used for specifying a distinct name for each informationType
737 element declared within a Choreography Package.

738 The OPTIONAL attributes type and element describe the type of information used
739 within a Choreography as a WSDL 1.1 Message Type, an XML Schema type, a
740 WSDL 2.0 Schema element or an XML Schema element. The type of information
741 is exclusively one of the aforementioned.

742 When the OPTIONAL attribute exceptionType is set to "true", this Information Type
743 is an *Exception Type* and MAY map to a WSDL fault type. When the attribute
744 exceptionType is set to "false", this information type MUST NOT map to a WSDL
745 fault type. The default for this attribute is set to "false".

746 In case of WSDL 2.0, the attribute element within the informationType refers to a
747 unique WSDL 2.0 faultname when the attribute exceptionType is set to "true".

748

749 The examples below show some possible usages of the informationType
750 construct.
751

752 **Example1:** The informationType "`purchaseOrder`" refers to the WSDL 1.1 Message type
753 "`pns:purchaseOrderMessage`"
754

```
755    <informationType name="purchaseOrder" type="pns:purchaseOrderMessage"/>
```

756
757
758 **Example2:** The informationType "`customerAddress`" refers to the WSDL 2.0 Schema element
759 "`cns:CustomerAddress`"
760

```
761   <informationType name="customerAddress" element="cns:CustomerAddress"/>
```

762
763
764 **Example 3:** The informationType "`intType`" refers to the XML Schema type "`xsd:int`"
765

```
766   <informationType name="intType" type="xsd:int"/>
```

767
768
769 **Example 4:** The informationType "`OutOfStockExceptionType`" is of type Exception Type and refers
770 to the WSDL 2.0 fault name "`cwns:OutOfStockExceptionType`"
771

```
772    <informationType name="OutOfStockExceptionType"
773                     type="cwns:OutOfStockExceptionType" exceptionType="true"/>
```

2.4.2 Variables

Variables capture information about objects in a Choreography as defined by their usage:

- *Information Exchange Capturing Variables*, which 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 Capturing Variables*, which contain information about the observable changes at a Role as a result of information being exchanged. For example when a Buyer sends an “Order” to a Seller, the Buyer could have a Variable called "OrderState" set to a value of "OrderSent" and once the message was received by the Seller, the Seller could have a 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
- *Channel Capturing Variables*. For example, a Channel Variable could contain information such as; the URL to which the message could be sent, the policies that are to be applied (e.g. security), whether or not reliable messaging is to be used, etc.

The value of Variables:

- Are available to Roles within a Choreography, when the Variables contain information that is common knowledge. For example the Variable "OrderResponseTime" which is the time in hours in which a response to an Order must be sent is initialized prior to the initiation of a Choreography and can be used by all Roles within the Choreography
- Can be made available as a result of an Interaction
  - Information Exchange Capturing Variables are populated and become available at the Roles at the ends of an Interaction
  - State Capturing Variables, that contain information about the observable information changes of a Role as a result of information being exchanged, are recorded and become available
- Can be created or changed and made available locally at a Role by assigning data from other information. They can be Information Exchange, State or Channel Capturing Variables. 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 ordering of the Choreography. In this case how “Maximum Order Amount” is calculated and its value would not be known by the other Roles

813 • Can be used to determine the decisions and actions to be taken within a
814 Choreography

815 • Can be used to cause Exceptions at one or more parties in a
816 Choreography

817 • Defined at different Roles that are part of the same Participant is shared
818 between these Roles when the Variables have the same name

819

820 The *variableDefinitions* construct is used for defining one or more Variables
821 within a Choreography.
822
823 The syntax of the *variableDefinitions* construct is:
824

```
825 <variableDefinitions>
826    <variable   name="ncname"
827        informationType="qname"?|channelType="qname"?
828        mutable="true|false"?
829        free="true|false"?
830        silent="true|false"?
831        roleTypes="list of qname"? />+
832 </variableDefinitions>
```

833

834 A Variable defined using the attribute informationType specifies either Information
835 Exchange Capturing Variables or State Capturing Variables. A Variable defined
836 using the attribute informationType specifies Exception Capturing Variables when
837 the referenced information type has the attribute exceptionType set to "true". A
838 Variable defined using the attribute channelType specifies Channel Capturing
839 Variables. The attributes informationType and channelType are mutually exclusive.

840 The OPTIONAL attribute mutable, when set to "false", specifies that the Variable
841 information cannot change once initialized. The default value for this attribute is
842 “true”.

843 The OPTIONAL attribute silent, when set to "true" specifies that there SHOULD
844 NOT be any activity used for creating or changing this Variable in the
845 Choreography. A silent Variable is used to represent the result of actions within a
846 party that are either not observable or are of no interest from the WS-CDL
847 perspective. The default value for this attribute is “false”.

848 The OPTIONAL attribute free, when set to "true" specifies that a Variable defined
849 in an enclosing Choreography is also used in this Choreography, thus sharing
850 the Variables information. The following rules apply in this case:

851 • The type (as specified by the informationType or the channelType attributes) of
852 a free Variable MUST match the type of the Variable defined in an
853 enclosing Choreography

854 • The attributes silent and mutable of a free Variable MUST match the
855 attributes silent and mutable of the Variable defined in an enclosing
856 Choreography

857 • A perform activity MUST bind a free Variable defined in an performed
858 Choreography with a Variable defined in a performing Choreography

859 The OPTIONAL attribute free, when set to "false" specifies that a Variable is
860 defined in this Choreography.
861 The default value for the free attribute is “false”.

862 The OPTIONAL attribute roleTypes is used to specify an XML-Schema list of one
863 or more Role Type(s) of a party at which the Variable information will reside. A
864 Variable defined without a Role Type is equivalent to a Variable that is defined at
865 all the Role Types that are part of the Relationship Types of the Choreography
866 where the Variable is defined. For example if Choreography “C1” has
867 Relationship Type “R” that has Roles “Role1”, “Role2”, then a Variable “var”
868 defined in Choreography “C1” without a roleTypes attribute means it is defined at
869 both “Role1” and “Role2”.

870 The attribute name is used for specifying a distinct name for each Variable
871 declared within the variableDefinitions element. In those cases where the visibility of
872 a Variable is wholly within a single Role then that Role needs to be named in the
873 definition of the Variable as the Role Type using the attribute roleTypes. In those
874 cases where the Variable is shared amongst a subset of Roles within a
875 Choreography those Roles need to be listed within the definition of the Variable
876 as the Role Types using the attribute roleTypes.

877 2.4.3 Expressions

878 Expressions can be used within WS-CDL to obtain existing information and to
879 create new or change existing information.

880 Generic expressions and literals can be used for populating a Variable. Predicate
881 expressions can be used within WS-CDL to specify conditions. Query
882 expressions are used within WS-CDL to specify query strings.

883 The language used in WS-CDL for specifying expressions and query or
884 conditional predicates is XPath 1.0.

885 WS-CDL defines XPath function extensions as described in the following section.
886 The function extensions are defined in the standard WS-CDL namespace
887 “http://www.w3.org/2004/12/ws-chor/cdl”. The prefix "cdl:" is associated with this
888 namespace.

889 2.4.3.1 WS-CDL Supplied Functions

890 There are several functions that the WS-CDL specification supplies as XPATH
891 1.0 extension functions. These functions can be used in any XPath expression as
892 long as the types are compatible:

893

894 *xsd:time getCurrentTime(xsd:QName roleName)*

895 Returns the current time at the caller for the Role specified by *roleName* (for
896 example a Role can ask only about it's own time).

897

898 *xsd:date getCurrentDate(xsd:QName roleName)*

899 Returns the current date at the caller for the Role specified by *roleName* (for
900 example a Role can ask only about it's own date).

901

902 *xsd:dateTime getCurrentDateTime(xsd:QName roleName)*

903 Returns the current date and time at the caller for the Role specified by *roleName*
904 (for example a Role can ask only about it's own date/time).

905

906 *xsd:boolean hasDurationPassed(xsd:duration elapsedTime, xsd:QName*
907 *roleName)*

908 Returns “true” if, (a) used in a guard or repetition condition of a Work Unit with
909 the block attribute set to “true” or in a complete condition of a Choreography and
910 (b) the duration specified by *elapsedTime* at the caller for the Role specified by
911 *roleName* has elapsed from the time either the guard or the repetition condition
912 were enabled for matching or the Choreography was enabled. Otherwise it
913 returns “false”.

914

915 *xsd:boolean hasDeadlinePassed(xsd:dateTime deadlineTime, xsd:QName*
916 *roleName)*

917 Returns "true" if, (a) used in a guard or repetition condition of a Work Unit with
918 the block attribute set to "true" or in a complete condition of a Choreography and
919 (b) the time specified by *deadlineTime* at the Role specified by *roleName* has
920 elapsed given that either the guard or the repetition condition were enabled for
921 matching or the Choreography was enabled. Otherwise it returns “false”.

922

923 *xsd:string createNewID()*

924 Returns a new globally unique value of XML-Schema ‘string’ type.

925

926 *xsd:any getVariable(xsd:string varName, xsd:string part, xsd:string*
927 *documentPath, xsd:QName roleName?)*

928 Returns the information of the Variable with name *varName* as a node set
929 containing a single node. The second parameter, *part*, specifies the message
930 part of a WSDL1.1 document. For a WSDL 2.0 document it MUST be empty.
931 When the third parameter *documentPath* is empty, then this function retrieves the
932 entire document from the Variable information. When it is non-empty, then this
933 function retrieves from the Variable information, the fragment of the document at
934 the provided absolute location path. The fourth parameter is OPTIONAL. When

935 the fourth parameter is used, the Variable information MUST be available at the
936 Role specified by *roleName.* If this parameter is not used then the Role is
937 inferred from the context that this function is used.

938

939 *xsd:boolean isVariableAvailable(xsd:string varName, xsd:QName roleName)*

940 Returns "true" if the information of the Variable with name *varName* is available
941 at the Role specified by *roleName*. Returns "false" otherwise.

942

943 *xsd:boolean variablesAligned(xsd:string varName, xsd:string withVarName,*
944 *xsd:QName relationshipName)*

945 Returns "true" if within a Relationship specified by *relationshipName* the Variable
946 with name *varName* residing at the first Role of the Relationship has aligned its
947 information with the Variable named *withVarName* residing at the second Role of
948 the Relationship.

949

950 *xsd:any getChannelReference(xsd:string varName)*

951 Returns the reference information of the Variable with name *varName*. The
952 Variable MUST be of Channel Type.

953

954 *xsd:any getChannelIdentity(xsd:string varName)*

955 Returns the identity information of the Variable with name *varName*. The Variable
956 MUST be of Channel Type.

957

958 *xsd:boolean globalizedTrigger(xsd:string expression, xsd:string roleName,*
959 *xsd:string expression2, xsd:string roleName2, …)*

960 Combines expressions that include Variables that are defined at different Roles.
961 Only one expression MUST be defined per Role name.
962

963 *xsd:boolean cdl:hasExceptionOccurred(xsd:string exceptionType)*

964 Returns "true" if an Exception of Exception Type identified by the parameter
965 *exceptionType* has occurred. Otherwise it returns "false".

966 ### 2.4.4 Tokens

967 A *Token* is an alias for a piece of data in a Variable or message that needs to be
968 used by a Choreography. Tokens differ from Variables in that Variables contain
969 values whereas Tokens contain information that define the piece of the data that
970 is relevant.

All Tokens MUST have an informationType, for example, an “Order Id” could be ‘alphanumeric’ and a “counter” an ‘integer’.

Tokens reference a document fragment within a Choreography definition and Token Locators provide a query mechanism to select them. By introducing these abstractions, a Choreography definition avoids depending on specific message types, as described by WSDL, or a specific query string, as specified by XPATH. Instead the document part and the query string can change without affecting the Choreography definition.

The syntax of the *token* construct is:

```
<token  name="ncname"  informationType="qname" />
```

The attribute name is used for specifying a distinct name for each token element declared within a Choreography Package.

The attribute informationType identifies the type of the document fragment.

The syntax of the *tokenLocator* construct is:

```
<tokenLocator  tokenName="qname"
               informationType="qname"
               part=”ncname”?
               query="XPath-expression" />
```

The attribute tokenName identifies the name of the Token that the document fragment locator is associated with.

The attribute informationType identifies the type of the document on which the query is performed to locate the Token.

The OPTIONAL attribute part defines the document part on which the query is performed to locate the Token. This attribute SHOULD NOT be defined for a WSDL 2.0 document.

The attribute query defines the query string that is used to select a document fragment within a document or a document part.

The example below shows that the Token “purchaseOrderID” is of XML-Schema type ‘int’. The two Token Locators show how to access this Token in the "purchaseOrder" and "purchaseOrderAck" messages.

```
<token name="purchaseOrderID" informationType="xsd:int"/>
```

```
              query="/PO/OrderId"/>
<tokenLocator tokenName="tns:purchaseOrderID" informationType="purchaseOrderAck"
              query="/POAck/OrderId"/>
```

2.4.5 Choreographies

A *Choreography* defines re-usable common rules, that govern the ordering of exhanged messages and the provisioning patterns of collaborative behavior, agreed between two or more interacting parties.

A Choreography defined at the Choreography Package level is called a *top-level* Choreography, and does not share its context with other top-level Choreographies. A Choreography Package MAY contain exactly one top-level Choreography, marked explicitly as the *root* Choreography. The root Choreography is the only top-level Choreography that is enabled by default.

The re-usable behavior encapsulated within a Choreography MAY be performed within an *enclosing* Choreography, thus facilitating composition. The performed Choreography is then called an *enclosed* Choreography and MAY be defined:

- *Locally* - its definition is contained within the enclosing Choreography
- *Globally* - a separate top-level, non-root Choreography definition is specified in the same or in a different Choreography Package that can be used by other Choreographies and hence the contract described becomes reusable

A non-root Choreography is enabled when performed.

A Choreography MUST contain at least one Relationship Type, enumerating the observable behavior this Choreography requires its parties to exhibit. One or more Relationship Types MAY be defined within a Choreography, modeling multi-party collaborations.

A Choreography acts as a lexical name scoping context for Variables. A Variable defined in a Choreography is visible for use in this Choreography and all its enclosed Choreographies up-to the point that the Variable is re-defined as an non-free Variable, thus forming a *Choreography Visibility Horizon* for this Variable.

A Choreography MAY contain one or more Choreography definitions that MAY be performed only locally within this Choreography.

A Choreography MUST contain an *Activity-Notation*. The Activity-Notation specifies the actions of the Choreography that perform the actual work. These actions are enabled when the Choreography they belong to is enabled.

A Choreography can recover from exceptional conditions by defining one *Exception Block*, which MAY be defined as part of the Choreography to recover from exceptional conditions that can occur.

An enclosed Choreography that has successfully completed MAY need to provide finalization actions that confirm, cancel or otherwise modify the effects of its completed actions. To handle these modifications, one or more separate Finalizer Block(s) MAY be defined for an enclosed Choreography.

A Choreography can also be coordinated. *Choreography Coordination* guarantees that all involved Roles agree on how the Choreography ended. That is, if the Choreography completed successfully or suffered an Exception, and if the Choreography completed successfully and Finalizer Block(s) were installed, all Roles have the same Finalizer Block enabled.

The *Choreography-Notation* is used to define a Choreography as follows:

```
<choreography  name="ncname"
      complete="xsd:boolean XPath-expression"?
      isolation="true"|"false"?
      root="true"|"false"?
      coordination="true"|"false"? >

   <relationship  type="qname" />+

   variableDefinitions?

   Choreography-Notation*

      Activity-Notation

   <exceptionBlock  name="ncname">
      WorkUnit-Notation+
   </exceptionBlock>?

   <finalizerBlock  name="ncname">
      WorkUnit-Notation
   </finalizerBlock>*

</choreography>
```

The attribute name is used for specifying a distinct name for each choreography element declared within a Choreography Package.

The OPTIONAL complete attribute allows to explicitly complete a Choreography as described below in the Choreography Life-line section.

The OPTIONAL isolation attribute specifies when a Variable defined in an enclosing Choreography, and changed within an enclosed Choreography is available to its sibling Choreographies. The default for this attribute is set to "false". The following rules apply:

- When isolation is set to "false", the Variable information MAY be immediately overwritten by actions in its sibling Choreographies

1095 • When isolation is set to "true", changes to the Variable information MUST
1096 be visible for read or for write to its sibling Choreographies only after this
1097 Choreography has completed

1098 The OPTIONAL coordination attribute specifies whether Choreography
1099 Coordination is required. The default for this attribute is set to "false". The
1100 following rules apply:

1101 • When the coordination attribute is set to "true", Choreography Coordination
1102 is required and a Coordination protocol MUST ensure that all the Roles
1103 agree on how the Choreography ended

1104 • When the coordination attribute is set to "false", the Choreography is not
1105 bound to a Coordination protocol, and since none of the above guarantees
1106 of agreement on the outcome apply any required coordination SHOULD
1107 be performed using explicitly modeled Interactions

1108 The relationship element within the choreography element enumerates the
1109 Relationships this Choreography MAY participate in.

1110 The OPTIONAL variableDefinitions element enumerates the Variables defined in
1111 this Choreography.

1112 The OPTIONAL root element marks a top-level Choreography as the root
1113 Choreography of a Choreography Package.

1114 The OPTIONAL Choreography-Notation within the choreography element defines
1115 the Locally defined Choreographies that MAY be performed only within this
1116 Choreography.

1117 The OPTIONAL exceptionBlock element defines the Exception Block of a
1118 Choreography by specifying one or more Exception Work Unit(s) using a
1119 *WorkUnit-Notation*. Within this element, the attribute name is used for specifying a
1120 name for this Exception Block element.

1121 The OPTIONAL finalizerBlock element defines a Finalizer Block for a
1122 Choreography. A Choreography MAY have more than one Finalizer Blocks. Each
1123 Finalizer Block specifies one Finalizer Work Unit using a *WorkUnit-Notation*. If a
1124 Choreography defines more than one Finalizer Blocks, then each MUST be
1125 differentiated by a distinct name as specified with the name attribute within the
1126 finalizerBlock element.

1127 2.4.6 WorkUnits

1128 A *Work Unit* can prescribe the constraints that have to be fulfilled for making
1129 progress and thus performing actual work within a Choreography. A Work Unit
1130 can also prescribe the constraints that preserve the consistency of the
1131 collaborations commonly performed between the parties. Using a Work Unit an
1132 application can recover from errors that are the result of abnormal actions and
1133 can also finalize successfully completed Choreographies that need further action,
1134 for example to confirm or logically roll back effects, or to close the Choreography

so that any defined “rollback” Work Unit will not be enabled. Examples of a Work Unit include:

- 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
- 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 constraint that identifies the error

The guard condition of a Work Unit, if specified, expresses the interest on one or more Variable information (that already exist or will become available in the future) being available under certain prescribed constraints. The Work Unit’s expressed interest MUST be matched for its enclosed actions to be enabled.

A Work Unit completes successfully when all its enclosed actions complete successfully.

A Work Unit that completes successfully MUST be considered again for matching (based on its guard condition), if its repetition condition evaluates to "true".

The *WorkUnit-Notation* is used to define a Work Unit as follows:

```
<workunit  name="ncname"
      guard="xsd:boolean XPath-expression"?
      repeat="xsd:boolean XPath-expression"?
      block="true|false"? >

      Activity-Notation
</workunit>
```

The attribute name is used for specifying a name for each Work Unit element declared within a Choreography Package.

The Activity-Notation specifies the enclosed actions within a Work Unit.

The OPTIONAL attribute guard specifies the guard condition of a Work Unit.

The OPTIONAL attribute repeat specifies the repetition condition of a Work Unit.

The OPTIONAL attribute block specifies whether the Work Unit has to block waiting for referenced Variables within the guard condition to become available (if they are not already) and the guard condition to evaluate to “true”. This attribute MUST always be set to “false” in Exception Work Units. The default for this attribute is set to “false”.

The following rules apply:

1175 • When a guard condition is not specified then the Work Unit always
1176 matches

1177 • One or more Work Units MAY be matched concurrently if their respective
1178 expressed interests are matched

1179 • When a repetition condition is not specified then the Work Unit is not
1180 considered again for matching after the Work Unit was matched once

1181 • One or more Variables can be specified in a guard condition or repetition
1182 condition, using XPATH and the WS-CDL functions, as described in
1183 Section 2.4.3.1

1184 • The WS-CDL function getVariable is used in the guard or repetition
1185 condition to obtain the information of a Variable

1186 • When the WS-CDL function isVariableAvailable is used in the guard or
1187 repetition condition, it means that the Work Unit that specifies the guard or
1188 repetition condition is checking if a Variable is already available at a
1189 specific Role or is waiting for a Variable to become available at a specific
1190 Role, based on the block attribute being "false" or "true" respectively

1191 • When the WS-CDL function variablesAligned is used in the guard or
1192 repetition condition, it means that the Work Unit that specifies the guard or
1193 repetition condition is checking or waiting for an appropriate alignment
1194 Interaction to happen between the two Roles, based on the block attribute
1195 being "false" or "true" respectively. The Variables checked or waited for
1196 alignment are the sending and receiving ones in an alignment Interaction
1197 or the ones used in the recordings at the two Roles at the ends of an
1198 alignment Interaction. When the variablesAligned WS-CDL function is used in
1199 a guard or repetition condition, then the Relationship Type within the
1200 variablesAligned MUST be the subset of the Relationship Type that the
1201 immediate enclosing Choreography defines

1202 • Variables defined at different Roles MAY be used in a guard condition or
1203 repetition condition to form a *globalized* view, thus combining constraints
1204 prescribed for each Role but without requiring that all these constraints
1205 have to be fullfilled for progress to be made. The globalizedTrigger WS-CDL
1206 function MUST be used in a guard condition or repetition condition in this
1207 case. Variables defined at the same Role MAY be combined together in a
1208 guard condition or repetition condition using all available XPATH operators
1209 and all the WS-CDL functions

1210 • If the attribute block is set to "true" and one or more required Variable(s)
1211 are not available or the guard condition evaluates to "false", then the Work
1212 Unit MUST block. When the required Variable information specified by the
1213 guard condition become available and the guard condition evaluates to
1214 "true", then the Work Unit is matched. If the repetition condition is
1215 specified, then it is evaluated when the Work Unit completes successfully.
1216 Then, if the required Variable information specified by the repetition
1217 condition is available and the repetition condition evaluates to "true", the

1218 Work Unit is considered again for matching. Otherwise, the Work Unit is
1219 not considered again for matching

1220 • If the attribute block is set to "false", then the guard condition or repetition
1221 condition assumes that the Variable information is currently available. If
1222 either the Variable information is not available or the guard condition
1223 evaluates to "false", then the Work Unit matching fails and the Activity-
1224 Notation enclosed within the Work Unit is skipped and the repetition
1225 condition is not evaluated even if specified. Otherwise, if the Work Unit
1226 matching succeeds, then the repetition condition, if specified, is evaluated
1227 when the Work Unit completes successfully. Then, if the required Variable
1228 information specified by the repetition condition is available and the
1229 repetition condition evaluates to "true", the Work Unit is considered again
1230 for matching. Otherwise, the Work Unit is not considered again for
1231 matching

1232

1233 The examples below demonstrate some usages of a Work Unit:

1234 *a. Example of a Work Unit with block equals to "true":*

1235 In the following Work Unit, the guard condition waits on the availability of
1236 “POAcknowledgement” at “Customer” Role and if it is already available, the
1237 activity happens, otherwise, the activity waits until the Variable
1238 “POAcknowledgement” become available at the “Customer” Role.
1239

```
1240 <workunit  name="POProcess"
1241            guard="cdl:isVariableAvailable(
1242                    cdl:getVariable("POAcknowledgement"), “”, “”, "tns:Customer")"
1243       block="true">
1244    ... <!--some activity -->
1245 </workunit>
```

1246

1247 *b. Example of a Work Unit with block equals to "false":*

1248 In the following Work Unit, the guard condition checks if the Variable
1249 “StockQuantity” at the “Retailer” Role is available and is greater than 10 and if so,
1250 the activity happens. If either the Variable is not available or its value is less than
1251 ‘10’, then the matching condition is "false" and the activity is skipped.
1252

```
1253 <workunit  name="StockCheck"
1254            guard="cdl:getVariable("StockQuantity", “”, "/Product/Qty",
1255                                   "tns:Retailer") > 10)"
1256            block="false" >
1257    ... <!--some activity -->
1258 </workunit>
```

1259

1260 *c. Example of a Work Unit waiting for alignment to happen:*

In the following Work Unit, the guard condition waits for an alignment Interaction to happen between the “Customer” Role and the “Retailer” Role:

```
<roleType name="Customer">
   …
</roleType>
<roleType name="Retailer">
   …
</roleType>
<relationshipType name="Customer-Retailer-Relationship">
    <role type="tns:Customer" />
    <role type="tns:Retailer" />
</relationshipType>

<workunit  name="WaitForAlignment"
           guard="cdl:variablesAligned(
                    "PurchaseOrderAtBuyer", "PurchaseOrderAtSeller",
                    "tns:Customer-Retailer-Relationship")”
       block="true" >
   ... <!--some activity -->
</workunit>
```

2.4.7 Choreography Life-line

A Choreography life-line expresses the progression of a collaboration through enabled activities and enclosed Choreographies. Initially, the collaboration is established between parties, then work is performed within it and finally it ends.

A Choreography is initiated, establishing a collaboration when an Interaction, explicitly marked as an *Choreography Initiator*, is performed. This causes the Exception Block to be installed and the Choreography enters the *Enabled State*. Before this point there is no observable association between any of the parties.

Two or more Interactions MAY be marked as Choreography Initiators, indicating alternatives for establishing a collaboration. In this case, the first performed Interaction will establish the collaboration and the other Interactions will enlist with the already established collaboration.

A Choreography Initiator Interaction MAY be defined within a root Choreography or within an enclosed Choreography. In either case the collaboration is established when the first Choreography Initiator Interaction is performed.

A Choreography in an Enabled State, completes unsuccessfully, when an Exception is caused in the Choreography and its Exception Block is enabled, if present. This causes the Choreography to enter the *Unsuccessfully Completed State*.

The unsuccessfully completed Choreography, enters the *Closed State* once the Exception Block, if present, is completed. If the Exception Block is not present, the Choreography implicitly enters the Closed State and the Exception occurred is propagated to the enclosing Choreography.

1307
1308 A Choreography in an Enabled State, completes successfully when there are no
1309 more enabled activities within its body. This causes its Exception Block to be
1310 deinstalled, Finalizer Blocks to be installed if specified, and the Choreography
1311 enters the *Successfully Completed State*.

1312 Alternatively, a Choreography completes successfully if its complete condition, is
1313 matched by evaluating to "true". A complete condition is considered for matching
1314 while the Choreography is in Enabled State. The complete condition MUST be
1315 possible to be matched in all Roles that participate in the Choreography. When
1316 the complete condition of a Choreography is matched then all activities in the
1317 Choreography are disabled, and the Choreography completes as if there were no
1318 more enabled activities within it. When a Choreography completes, all
1319 uncompleted enclosed Choreographies will automatically become completed.
1320 Messages that were sent as part of a Choreography that has since completed
1321 MUST be ignored.
1322
1323 A Choreography, in a Successfully Completed State, enters the Closed State if
1324 no Finalizer Blocks were specified in that Choreography.

1325 A Choreography, in a Successfully Completed State with Finalzer Block(s)
1326 specified enters the Closed State when one of its installed Finalizer Block(s) is
1327 enabled and completed. The Finalizer Block of a Choreography is enabled by a
1328 finalize activity in the immediately enclosing Choreography. Alternatively, a
1329 Choreography in Successfully Completed State with Finalzer Block(s) specified
1330 implicitly enters the Closed State when its enclosing Choreography enters the
1331 Closed State without enabling the Finalizer Block(s) of its enclosed
1332 Choreography. In other words, when a Choreography enters the Closed State, all
1333 its enclosed successfully completed Choreographies are implicitly entering the
1334 Closed State even if none of their Finalizer Blocks has been enabled.

1335 2.4.8 Choreography Exception Handling

1336 A Choreography can sometimes fail as a result of an exceptional circumstance or
1337 an “error” that occurred during its performance.

1338 An *Exception* is caused in the Choreography when an Exception Variable is
1339 populated in an Interaction activity with the attribute causeException set to “true”.

1340 An Exception MUST be propagated to all parties in the Choreography using
1341 explicitly modeled, *Exception Causing Interactions* when the Choreography is not
1342 coordinated. This causes the Choreography to enter the Exception state and its
1343 Exception Block to be enabled, if specified.

1344

1345 Different types of errors are possible including this non-exhaustive list:

1346 • *Interaction Failures*, for example the sending of a message did not
1347 succeed

1348 • *Protocol Based Exchange failures*, for example no acknowledgement was
1349 received as part of a reliable messaging protocol

1350 • *Security failures*, for example a Message was rejected by a recipient
1351 because the digital signature was not valid

1352 • *Timeout errors*, for example an Interaction did not complete within the
1353 required time

1354 • *Validation Errors*, for example an XML "Order" document was not well
1355 formed or did not conform to its XML-Schema definition

1356 • *Application "failures"*, for example the "goods ordered" were 'out of stock'

1357 To handle these and other "errors" separate *Exception Work Units* MAY be
1358 defined in the Exception Block of a Choreography, for each Exception that needs
1359 to be handled.

1360 One or more Exception Work Unit(s) MAY be defined within the Exception Block
1361 of a Choreography. At least one Exception Work Unit MUST be defined as part of
1362 the Exception Block of a Choreography. An Exception Work Unit MAY express
1363 interest on Exception information using its guard condition on Exception Types or
1364 Exception Variables. If no guard condition is specified, then the Exception Work
1365 Unit is called the *Default Exception Work Unit* and expresses interest on any type
1366 of Exception. Within the Exception Block of a Choreography there MUST NOT be
1367 more than one Default Exception Work Unit. An Exception Work Unit MUST
1368 always set its block attribute to "false" and MUST NOT define a repetition
1369 condition.

1370 Exception Work Units are enabled when the Exception Block of the
1371 Choreography they belong to is enabled. Enabled Exception Work Units in a
1372 Choreography MAY behave as the mechanism to recover from Exceptions
1373 occuring in this and its enclosed Choreographies.

1374 Within the Exception Block of a Choreography only one Exception Work Unit
1375 MAY be matched.

1376

1377 The rules for matching an Exception are:

1378 • When an Exception Work Unit has a guard condition using the
1379 hasExceptionOccurred(exceptionType) WS-CDL function, then it is matched
1380 when an Exception Variable with Exception Type that matches the
1381 parameter exceptionType is populated using an Exception Causing
1382 Interaction activity

1383 • If an Exception is matched by the guard condition of an Exception Work
1384 Unit, then the actions of the matched Work Unit are enabled. When two or
1385 more Exception Work Units are defined then the order of evaluating their
1386 guard conditions is based on the order that the Work Units have been
1387 defined within the Exception Block

1388 • If none of the guard condition(s) match, then if there is a Default Exception
1389 Work Unit without a guard condition defined then its actions are enabled

1390 • If an Exception is not matched by an Exception Work Unit defined within
1391 the Choreography in which the Exception occurs, the Exception will be
1392 recursively propagated to the Exception Work Unit of the immediate
1393 enclosing Choreography until a match is successful

1394 • If an Exception occurs within a Choreography, then the Choreography
1395 completes unsuccessfully. In this case its Finalizer Block(s) MUST NOT
1396 be installed. The actions, including enclosed Choreographies, within this
1397 Choreography are completed abnormally before an Exception Work Unit
1398 can be matched

1399 The actions within the Exception Work Unit MAY use Variable information visible
1400 in the Visibility Horizon of the Choreography it belongs to as they stand at the
1401 current time.

1402 The actions of an Exception Work Unit MAY also cause an Exception. The
1403 semantics for matching the Exception and acting on it are the same as described
1404 in this section.

1405 2.4.9 Choreography Finalization

1406 After a Choreography instance has successfully completed, it MAY need to
1407 provide finalization actions that confirm, cancel or otherwise modify the effects of
1408 its completed actions. To handle these modifications, one or more separate
1409 Finalizer Block(s) MAY be defined for an enclosed Choreography. When its
1410 Choreography body completes successfully, any Finalizer Blocks specified in the
1411 Choreography are installed.

1412 If more than one Finalizer Blocks are defined for the same Choreography, each
1413 of them MUST be differentiated by their name attributes. However, at most one
1414 Finalizer Block MAY be enabled for any given Choreography instance during the
1415 subsequent progress, including Exception handling and finalization, of the
1416 enclosing Choreography.

1417 Finalizer Block(s) MAY implement whatever actions are appropriate for the
1418 particular Choreography. Common patterns might include:

1419 • A single Finalizer Block to semantically "rollback" the Choreography

1420 • Two Finalizer Blocks, for example one with name "confirm" and one with
1421 name "cancel", to implement a two-phase outcome protocol

1422 • One "undo" Finalizer Block along with a "close" Finalizer Block to signal
1423 that the "undo" Finalizer Block is no longer able to be enabled, that is, the
1424 Choreography is now closed

1425 The actions within the Finalizer Work Unit MAY use Variable information visible
1426 in the Visibility Horizon of the Choreography it belongs to as they were at the
1427 time the Choreography completed for the Variables belonging to this

1428 Choreography and as they stand at the current time for the Variables belonging
1429 to the enclosing Choreography.

1430 The actions of a Finalizer Work Unit MAY fault. The semantics for matching the
1431 fault and acting on it are the same as described in the previous section.

1432 2.4.10 Choreography Coordination

1433 Choreography Coordination guarantees that all involved Roles will agree on how
1434 the Choreography ended. That is, all Roles will agree on whether the
1435 Choreography completed successfully or suffered an Exception, and if the
1436 Choreography completed successfully and Finalizer Blocks were installed, all
1437 Roles will agree on which Finalizer Block was enabled. Such agreement differs
1438 from Interaction based alignment in that the Choreography as a whole is aligned,
1439 regardless of whether each Interaction in the Coordinated Choreography is
1440 aligned. In contrast to Alignment Interactions, a Coordinated Choreography
1441 provides a larger unit of coordination - a set of Interactions that end with shared
1442 knowledge among all the parties that their Relationship is in a defined state.
1443 Such a unit need not be aligned at each step - it is only required that clear
1444 alignment points are made to guarantee that all involved Roles will agree on how
1445 the Choreography ended.

1446 Choreographies defined as requiring coordination must being bound to a
1447 Coordination protocol. When Choreography Coordination is not required, then
1448 the Choreography is not bound to a Coordination protocol, and since none of the
1449 above guarantees of agreement on the outcome apply any required coordination
1450 should be performed using explicitly modeled Interactions.

1451

1452 The implications of Choreography Coordination differ for root Choreographies
1453 versus enclosed Choreographies:

1454 • An enclosed Choreography MAY have one or more Finalizer Block(s). In
1455 this case, coordination means that all Roles agree on whether the
1456 Choreography completed successfully or suffered an Exception, and if the
1457 Choreography completed successfully and Finalizer Block(s) were
1458 installed, all Roles agree on which Finalizer Block was enabled

1459 • A root Choreography can also be coordinated, but it cannot have any
1460 Finalizer Block(s). In this case, coordination means that all Roles agree on
1461 whether the Choreography completed successfully or suffered an
1462 Exception

1463 • In both cases, all Roles MUST agree on whether the Choreography
1464 completed successfully, or if an Exception occurs, all Roles MUST
1465 experience an Exception rather than successful completion. When an
1466 Exception occurs within a Choreography, the Coordination protocol will
1467 throw an Exception to Roles which have not otherwise detected the
1468 Exception that occurred

1469

1470 The two examples below show two usages of Coordinated Choreographies.

1471

```
1472  Example 1: Coordinated credit authorization without Finalizer Block(s):
1473
1474  <informationType name="creditDeniedType" exceptionType="true"/>
1475
1476  <!-- Coordinated CreditAuthorization  choreography without Finalizer Block(s)-->
1477  <choreography name="CreditAuthorization" root="false" coordination="true">
1478     <relationship type="tns:CreditReqCreditResp"/>
1479     <variableDefinitions>
1480        <variable name="CreditExtended" informationType="xsd:int" silent="true"
1481                  roleTypes="tns:CreditResponder"/>
1482        <variable name="creditRequest"/>
1483        <variable name="creditAuthorized"/>
1484        <variable name="creditDenied" informationType = "creditDeniedType"/>
1485     </variableDefinitions>
1486
1487     <!-- the normal work - receive the request and decide whether to approve -->
1488     <interaction name="creditAuthorization" channelVariable="tns:CreditRequestor"
1489                  operation="authorize">
1490        <participate relationshipType="SuperiorInferior" fromRole="tns:Superior"
1491                     toRole="Inferior"/>
1492        <exchange name="creditRequest" informationType="creditRequest"
1493                  action="request">
1494           <send variable="tns:creditRequest"/>
1495           <receive variable="tns:creditRequest"/>
1496        </exchange>
1497        <exchange name="creditAuthorized" informationType="creditDenied"
1498                  action="respond">
1499           <send variable="tns:creditAuthorized"/>
1500           <receive variable="tns:creditAuthorized"/>
1501        </exchange>
1502        <exchange name="creditDenied" informationType="refusal" action="respond">
1503           <send variable="tns:creditDenied" causeException="true"/>
1504           <receive variable="tns:creditDenied" causeException="true"/>
1505        </exchange>
1506     </interaction>
1507
1508     <!-- catch the (application) exception - as an exception it will abort the
1509         choreography -->
1510     <exceptionBlock name="handleBadCreditException">
1511       <workunit  name="handleBadCredit" >
1512                 <interaction name="badCreditInteraction"
1513                              channelVariable="tns:CreditResponder"
1514                              operation="creditDenied">
1515                     <participate relationshipType="CreditReqCreditResp"
1516                          fromRole="tns:Responder" toRole="CreditRequestor"/>
1517        </interaction>
1518       </workunit>
1519    </exceptionBlock>
1520  </choreography>
1521
1522
1523  Example 2: Coordinated credit authorization with Finalizer Block(s):
1524
1525  <informationType name="creditDeniedType" exceptionType="true"/>
1526
1527  <!-- Coordinated CreditAuthorization  choreography with Finalizer Block(s) -->
1528  <choreography name="CreditAuthorization" root="false" coordination="true">
```

```
1529
1530    <variableDefinitions>
1531       <variable name="CreditExtended" informationType="xsd:int" silent="true"
1532                 roleTypes="tns:CreditResponder"/>
1533       <variable name="creditRequest"/>
1534       <variable name="creditAuthorized"/>
1535       <variable name="creditDenied" informationType = "creditDeniedType"/>
1536    </variableDefinitions>
1537
1538    <!-- the normal work -receive the request and decide whether to approve -->
1539    <interaction name="creditAuthorization" channelVariable="tns:CreditRequestor"
1540                 operation="authorize">
1541       <participate relationshipType="SuperiorInferior" fromRole="tns:Superior"
1542                    toRole="Inferior"/>
1543       <exchange name="creditRequest" informationType="creditRequest"
1544              action="request">
1545          <send variable="tns:creditRequest"/>
1546          <receive variable="tns:creditRequest"/>
1547       </exchange>
1548       <exchange name="creditAuthorized" informationType="creditDenied"
1549                 action="respond">
1550          <send variable="tns:creditAuthorized"/>
1551          <receive variable="tns:creditAuthorized"/>
1552       </exchange>
1553       <exchange name="creditDenied" informationType="refusal" action="respond">
1554          <send variable="tns:creditDenied" causeException="true"/>
1555          <receive variable="tns:creditDenied" causeException="true"/>
1556       </exchange>
1557    </interaction>
1558
1559    <!-- catch the (application) exception - as an exception it will abort the
1560         choreography and the Finalizer Block(s) are not accessible -->
1561    <exceptionBlock name="handleBadCreditException">
1562        <workunit  name="handleBadCredit" >
1563           <interaction name="badCreditInteraction"
1564                 channelVariable="tns:CreditResponder"
1565                 operation="creditDenied">
1566                 <participate relationshipType="CreditReqCreditResp"
1567                       fromRole="tns:Responder" toRole="CreditRequestor"/>
1568           </interaction>
1569        </workunit>
1570    </exceptionBlock>
1571
1572    <!-- Finalizer Block(s) -->
1573    <!-- what to do if the credit is drawn down -->
1574    <finalizerBlock name="drawDown">
1575       <!-- if there is no application content to send, this could just be an
1576            assignment to the statecapturevariable creditExtended -->
1577        <workunit  name="drawdown" >
1578             <interaction name="drawdownInteraction"
1579                          channelVariable="tns:CreditRequestor"
1580                          operation="drawDown">
1581                   <participate relationshipType="CreditReqCreditResp"
1582                       fromRole="tns:CreditRequestor" toRole="CreditResponder"/>
1583                   <exchange name="dummy" action="request">
1584                       <send></send>
1585                       <receive recordReference="drawdownRecord"/>
1586                   </exchange>
1587                  <record name="drawdownRecord" when="before">
1588                       <source expression="drawnDown"/>
1589                       <target variable="CreditExtended"/>
1590                   </record>
1591             </interaction>
```

```
    </finalizerBlock>

    <!-- what to do if the credit is not used -->
    <finalizerBlock name="replenish">
    <!-- if there is no application content to send, this could just be an
            assignment to the state capturing variable creditExtended -->
        <workunit name="replenishWU">
             <interaction name="replenishInteraction"
                          channelVariable="tns:CreditRequestor"
                          operation="replenish">
                 <participate relationshipType="CreditReqCreditResp"
                       fromRole="tns:CreditRequestor" toRole="CreditResponder"/>
                 <exchange name="dummy" action="request">
                      <send></send>
                        <receive recordReference="replenishRecord"/>
                 </exchange>
                 <record name="replenishRecord" when="before">
                        <source expression="released"/>
                        <target variable="CreditExtended"/>
                 </record>
              </interaction>
         </workunit>
    </finalizerBlock>
</choreography>
```

2.5 Activities

Activities are the lowest level components of the Choreography, used to describe the actual work performed.

The *Activity-Notation* is used to define activities as either:

- An *Ordering Structure* – which combines Activities with other Ordering Structures in a nested way to specify the ordering rules of activities within the Choreography
- A *WorkUnit-Notation*
- A *Basic Activity* that performs the actual work. A Basic Activity is then either:
    - An *Interaction Activity*, which results in an exchange of information between parties and possible synchronization of their observable information changes and the actual values of the exchanged information
    - A *Perform Activity,* which means that a complete, separately defined Choreography is performed
    - An *Assign Activity*, which assigns, within one Role, the value of one Variable to another Variable
    - A *Silent Action Activity*, which provides an explicit designator used for specifying the point where party specific action(s) with non-observable operational details MUST be performed

  - A *No Action Activity*, which provides an explicit designator used for specifying the point where a party does not perform any action
  - A *Finalize Activity*, which enables a particular Finalizer Block in a particular instance of an immediately enclosed Choreography and thus brings that Choreography to a defined conclusion

2.5.1 Ordering Structures

An *Ordering Structure* is one of the following:

- *Sequence*
- *Parallel*
- *Choice*

2.5.1.1 Sequence

The *sequence* ordering structure contains one or more Activity-Notations. When the sequence activity is enabled, the sequence element restricts the series of enclosed activities (as defined by one or more Activity-Notations) to be enabled sequentially, in the same order that they are defined.

The syntax of this construct is:

```
<sequence>
    Activity-Notation+
</sequence>
```

2.5.1.2 Parallel

The *parallel* ordering structure contains one or more Activity-Notation that are enabled concurrently when the parallel activity is enabled. The parallel activity completes successfully when all activities (as defined by one or more Activity-Notations) performing work within it complete successfully.

The syntax of this construct is:

```
<parallel>
    Activity-Notation+
</parallel>
```

2.5.1.3 Choice

The *choice* ordering structure enables specifying that only one of two or more activities (as defined by two or more Activity-Notations) SHOULD be performed.

When two or more activities are specified in a choice element, only one activity is selected and the other activities are disabled. If the choice has Work Units with guard conditions, the first Work Unit that matches the guard condition is selected

and the other Work Units are disabled. If the choice has other activities, it is assumed that the selection criteria for the activities are non-observable.

The syntax of this construct is:

```
<choice>
    Activity-Notation+
</choice>
```

In the example below, choice element has two Interactions, “processGoodCredit” and “processBadCredit”. The Interactions have the same directionality, participate within the same Relationship and have the same fromRoles and toRoles names. If one Interaction happens, then the other one is disabled.

```
<choice>
  <interaction name=""processGoodCredit"
       channelVariable="goodCredit-channel" operation="doCredit">
      ...
  </interaction>

  <interaction name=""processBadCredit"
       channelVariable="badCredit-channel" operation="doBadCredit">
      ...
  </interaction>
<choice>
```

2.5.2 Interacting

An *Interaction* is the basic building block of a Choreography, which results in information exchanged between collaborating parties and possibly the synchronization of their observable information changes and the values of the exchanged information.

An Interaction forms the base atom of the Choreography composition, where multiple Interactions are combined to form a Choreography, which can then be used in different business contexts.

An Interaction is initiated when one of the Roles participating in the Interaction sends a message, through a common Channel, to another Role that is participating in the Interaction, that receives the message. If the initial message is a request, then the accepting Role can optionally respond with a normal response message or a fault message, which will be received by the initiating Role.

An Interaction also contains "references" to:

- The *Channel Capturing Variable* that specifies the interface and other data that describe where and how the message is to be sent to and received into the accepting Role

1720 • The *Operation* that specifies what the recipient of the message should do
1721 with the message when it is received

1722 • The *From Role* and *To Role* that are involved

1723 • The *Information Type or Channel Type* that is being exchanged

1724 • The *Information Exchange Capturing Variables* at the From Role and To
1725 Role that are the source and destination for the message content

1726 • A list of potential observable information changes that can occur and may
1727 need to be aligned at the From Role and the To Role, as a result of
1728 carrying out the Interaction

1729 #### 2.5.2.1 Interaction Based Information Alignment

1730 In some Choreographies there may be a requirement that, when the Interaction
1731 is performed, the Roles in the Choreography have agreement on the outcome.
1732 More specifically within an Interaction, a Role MAY need to have a common
1733 understanding of the observable information creations or changes of one or more
1734 *State Capturing Variables* that are complementary to one or more *State*
1735 *Capturing Variables* of its partner Role. Additionally, within an Interaction a Role
1736 MAY need to have a common understanding of the values of the Information
1737 Exchange Capturing Variables at the partner Role.

1738 For example, after an Interaction happens, both the Buyer and the Seller want to
1739 have a common understanding that:

1740 • State Capturing Variables, such as "Order State", that contain observable
1741 information at the Buyer and Seller, have values that are complementary
1742 to each other, e.g. “Sent” at the Buyer and “Received” at the Seller, and

1743 • Information Exchange Capturing Variables have the same types with the
1744 same content, e.g. The “Order” Variables at the Buyer and Seller have the
1745 same Information Types and hold the same order information

1746

1747 In WS-CDL, an *Alignment Interaction* MUST be explicitly used, in the cases
1748 where two interacting parties require the alignment of their observable
1749 information changes and the values of their exchanged information. After the
1750 alignment Interaction completes, both parties progress at the same time, in a
1751 lock-step fashion and the Variable information in both parties is aligned. Their
1752 Variable alignment comes from the fact that the requesting party has to know that
1753 the accepting party has received the message and the other way around, the
1754 accepting party has to know that the requesting party has sent the message
1755 before both of them progress. There is no intermediate state, where one party
1756 sends a message and then it proceeds independently or the other party receives
1757 a message and then it proceeds independently.

1758 #### 2.5.2.2 Interaction Life-line

1759 An Interaction completes normally when its message exchange(s) complete
1760 successfully.

1761 An Interaction completes abnormally when:

1762-1763
- An application signals an error condition during the management of a request or within a party when processing the request

1764-1766
- The *time-to-complete* timeout, identifying the timeframe within which an Interaction MUST complete, occurs after the Interaction was initiated but before it completed

1767-1768
- Other types of errors, such as Protocol Based Exchange failures, Security failures, Document Validation errors, etc.

1769
2.5.2.3 Interaction Syntax

1770 The syntax of the *interaction* construct is:

1771

```
1772 <interaction  name="ncname"
1773              channelVariable="qname"
1774              operation="ncname"
1775              align="true"|"false"?
1776              initiate="true"|"false"? >
1777
1778    <participate  relationshipType="qname"
1779                  fromRole="qname" toRole="qname" />
1780
1781    <exchange  name="ncname"
1782              informationType="qname"?|channelType="qname"?
1783              action="request"|"respond" >
1784      <send      variable="XPath-expression"?
1785                 recordReference="list of ncname"?
1786                 causeException="true"|"false"? />
1787      <receive   variable="XPath-expression"?
1788                 recordReference="list of ncname"?
1789                 causeException="true"|"false"? />
1790    </exchange>*
1791
1792    <timeout  time-to-complete="XPath-expression"
1793              fromRoleRecordReference="list of ncname"?
1794              toRoleRecordReference="list of ncname"? />?
1795
1796    <record  name="ncname"
1797             when="before"|"after"|"timeout"
1798             causeException="true"|"false"? >
1799      <source  variable="XPath-expression"? | expression="Xpath-expression"? />
1800      <target  variable="XPath-expression" />
1801    </record>*
1802 </interaction>
```

1803

1804-1805 The attribute name is used for specifying a name for each Interaction element declared within a Choreography.

1806-1810 The channelVariable attribute specifies the Channel Variable containing information of a party, being the target of the Interaction, which is used for determining where and how to send and receive information to and into the party. The Channel Variable used in an Interaction MUST be available at the two Roles before the Interaction occurs. At runtime, information about a Channel Variable is expanded

1811 further. This requires that the messages exchanged in the Choreography also
1812 contain reference and correlation information, for example by:

1813 • Including a protocol header, such as a SOAP header or

1814 • Using the actual value of data within a message, for example the “Order
1815 Number” of the Order that is common to all the messages sent over the
1816 Channel

1817 The operation attribute specifies the name of the operation that is associated with
1818 this Interaction. The specified operation belongs to the interface, as identified by
1819 the role and behavior elements of the Channel Type of the Channel Variable used
1820 in this Interaction.

1821 The OPTIONAL align attribute when set to "true" means that this Alignment
1822 Interaction results in the common understanding of both the information
1823 exchanged and the resulting observable information creations or changes at the
1824 ends of the Interaction as specified in the fromRole and the toRole. The default for
1825 this attribute is "false".

1826 An Interaction activity can be marked as a Choreography Initiator when the
1827 OPTIONAL initiate attribute is set to "true". The default for this attribute is "false".

1828 Within the participate element, the relationshipType attribute specifies the
1829 Relationship Type this Interaction participates in and the fromRole and toRole
1830 attributes specify the requesting and the accepting Role Types respectively. The
1831 Role Type identified by the toRole attribute MUST be the same as the Role Type
1832 identified by the role element of the Channel Type of the Channel Variable used in
1833 the interaction activity.

1834

1835 The OPTIONAL exchange element allows information to be exchanged during an
1836 Interaction. The attribute name is used for specifying a name for this exchange
1837 element.

1838 Within the exchange element, the OPTIONAL attributes informationType and
1839 channelType identify the Information Type or the Channel Type of the information
1840 that is exchanged between the two Roles in an Interaction. The attributes
1841 informationType and channelType are mutually exclusive. If none of these attributes
1842 are specified, then it is assumed that either no actual information is exchanged or
1843 the type of information being exchanged is of no interest to the Choreography
1844 definition.

1845 Within the exchange element, the attribute action specifies the direction of the
1846 information exchanged in the Interaction:

1847 • When the action attribute is set to “request”, then the information exchange
1848 happens fromRole to toRole

1849 • When the action attribute is set to ”respond”, then the information exchange
1850 happens from toRole to fromRole

1851 Within the exchange element, the send element shows that information is sent from
1852 a Role and the receive element shows that information is received at a Role
1853 respectively in the Interaction:

1854
1855
- The send and the receive elements MUST only use the WS-CDL function getVariable within the variable attribute

1856
1857
- The OPTIONAL Variables specified within the send and receive elements MUST be of type as described in the informationType or channelType attributes

1858
1859
1860
1861
- When the action element is set to "request", then the Variable specified within the send element using the variable attribute MUST be defined at the fromRole and the Variable specified within the receive element using the variable attribute MUST be defined at the toRole

1862
1863
1864
1865
- When the action element is set to "respond", then the Variable specified within the send element using the variable attribute MUST be defined at the toRole and the Variable specified within the receive element using the variable attribute MUST be defined at fromRole

1866
1867
- The Variable specified within the receive element MUST not be defined with the attribute silent set to “true”

1868
1869
1870
1871
1872
- Within the send or the receive element(s) of an exchange element, the recordReference attribute contains an XML-Schema list of references to record element(s) in the same Interaction. The same record element MAY be referenced from different send or the receive element(s) within the same Interaction thus enabling re-use

1873
1874
1875
1876
1877
- Within the send or the receive element(s) of an exchange element, the causeException attribute when set to “true”, specifies that an Exception MAY be caused at the respective Roles. In this case, the informationType of the exchange element MUST be of Exception Type. The default for this attribute is "false"

1878
1879
- The request exchange MUST NOT have causeException attribute set to “true”

1880
1881
1882
1883
- When two or more respond exchanges are specified, one respond exchange MAY be of normal informationType and all others MUST be of Exception Type. There is an implicit choice between two or more respond exchanges

1884
1885
- If the align attribute is set to "false" for the Interaction, then it means that the:

1886
1887
1888
1889
1890
1891
  - Request exchange completes successfully for the requesting Role once it has successfully sent the information of the Variable specified within the send element and the Request exchange completes successfully for the accepting Role once it has successfully received the information of the Variable specified within the receive element

1892 o Response exchange completes successfully for the accepting Role
1893 once it has successfully sent the information of the Variable
1894 specified within the send element and the Response exchange
1895 completes successfully for the requesting Role once it has
1896 successfully received the information of the Variable specified
1897 within the receive element

1898 • If the align attribute is set to "true" for the Interaction, then it means that the
1899 Interaction completes successfully if its Request and Response
1900 exchanges complete successfully and all referenced records complete
1901 successfully:

1902 o A Request exchange completes successfully once both the
1903 requesting Role has successfully sent the information of the
1904 Variable specified within the send element and the accepting Role
1905 has successfully received the information of the Variable specified
1906 within the receive element

1907 o A Response exchange completes successfully once both the
1908 accepting Role has successfully sent the information of the Variable
1909 specified within the send element and the requesting Role has
1910 successfully received the information of the Variable specified
1911 within the receive element

1912

1913 Within the OPTIONAL timeout element, the time-to-complete attribute identifies the
1914 timeframe within which an Interaction MUST complete after it was initiated or the
1915 deadline before an Interaction MUST complete. The time-to-complete SHOULD be
1916 of XML-Schema duration type when conveying the timeframe and SHOULD be of
1917 XML-Schema dateTime type when conveying the deadline. The OPTIONAL
1918 fromRoleRecordReference attribute contains an XML-Schema list of references to
1919 record element(s) in the same Interaction that will take effect at the fromRole
1920 when a timeout occurs. The OPTIONAL toRoleRecordReference attribute contains
1921 an XML-Schema list of references to record element(s) in the same Interaction
1922 that will take effect at the toRole when a timeout occurs.

1923

1924 The OPTIONAL element record is used to create or change and then make
1925 available within one Role, the value of one or more Variables using another
1926 Variable or an expression. The attribute name is used for specifying a distinct
1927 name for a record element within an Interaction. Within the record element, the
1928 source and target elements specify these recordings of information happening at
1929 the send and receive ends of the Interaction:

1930 • When the action element is set to "request", then the recording(s) specified
1931 within the source and the target elements occur at the fromRole for the send
1932 and at the toRole for the receive

1933 • When the action element is set to "response", then the recording(s)
1934 specified within the source and the target elements occur at the toRole for
1935 the send and at the fromRole for the receive

1936 Within the record element, the when attribute specifies if a recording happens
1937 before or after a send or “before” or “after” a receive of a message at a Role in a
1938 Request or a Response exchange or when a timeout has expired. When the when
1939 attribute is set to “timeout”, the record element specifies the recording to be
1940 performed when a timeout occurs. If two or more record elements have the same
1941 value in their when attribute and are referenced within the recordReference attribute
1942 of a send or a receive element, then they are performed in the order in which they
1943 are specified.

1944 The following rules apply for the information recordings when using the record
1945 element:

1946 • The source MUST define either a variable attribute or an expression attribute:

1947 o When the source defines an expression attribute, it MUST contain
1948 expressions, as defined in Section 2.4.3. The resulting type of the
1949 defined expression MUST be compatible with the target Variable
1950 type

1951 o When the source defines a Variable, then the source and the target
1952 Variable MUST be of compatible type

1953 o When the source defines a Variable, then the source and the target
1954 Variable MUST be defined at the same Role

1955 • When the attribute variable is defined it MUST use only the WS-CDL
1956 function getVariable

1957 • The target Variable MUST NOT be defined with the attribute silent set to
1958 “true”

1959 • One or more record elements MAY be specified and performed at one or
1960 both the Roles within an Interaction

1961 • A record element MUST NOT be specified in the absence of an exchange
1962 element or a timeout element that reference it

1963 • The attribute causeException MAY be set to "true" in a record element if the
1964 target Variable is an Exception Variable

1965 • When the attribute causeException is set to "true" in a record element, the
1966 corresponding Role gets into the Exception state

1967 • When two or more record elements are specified for the same Role in an
1968 Interaction with target Variables of Exception Type, one of the Exception
1969 recordings MAY occur. An Exception recording has an non-observable
1970 predicate condition, associated implicitly with it, that decides if an
1971 Exception occurs

1972 • If the align attribute is set to "false" for the Interaction, then it means that
1973 the Role specified within the record element makes available the creation
1974 or change of the information specified within the record element
1975 immediately after the successful completion of each record

1976 • If the align attribute is set to "true" for the Interaction, then it means that

1977 o Both Roles know the availability of the creation or change of the
1978 information specified within the record element only at the successful
1979 completion of the Interaction

1980 o If there are two or more record elements specified within an
1981 Interaction, then all record operations MUST complete successfully
1982 for the Interact to complete successfully. Otherwise, none of the
1983 Variables specified in the target attribute will be affected

1984

1985 The example below shows a complete Choreography that involves one
1986 Interaction performed from Role Type “Consumer” to Role Type “Retailer” on the
1987 Channel "retailer-channel" as a request/response exchange:

1988 • The message “purchaseOrder” is sent from the “Consumer” to the
1989 “Retailer” as a request message

1990 • The message “purchaseOrderAck” is sent from the “Retailer” to the
1991 “Consumer” as a response message

1992 • The Variable “consumer-channel” is made available at the “Retailer” using
1993 the record element

1994 • The Interaction happens on the “retailer-channel”, which has a Token
1995 “purchaseOrderID” used within the identity element of the Channel. This
1996 identity element is used to identify the business process of the “Retailer”

1997 • The request message “purchaseOrder” contains the identity of the
1998 “Retailer” business process as specified in the tokenLocator for
1999 ”purchaseOrder” message

2000 • The response message “purchaseOrderAck” contains the identity of the
2001 “Consumer” business process as specified in the tokenLocator for
2002 “purchaseOrderAck” message

2003 • The “consumer-channel” is sent as a part of “purchaseOrder” Interaction
2004 from the “Consumer” to the “Retailer” on “retailer-channel” during the
2005 request. Here the record element makes available the “Consumer-channel”
2006 at the “Retailer” Role. If the align attribute was set to "true" for this
2007 Interaction, then it also means that the “Consumer” knows that the
2008 “Retailer” now has the contact information of the “Consumer”. In another
2009 example, the “Consumer” could set its Variable "OrderSent" to "true" and
2010 the “Retailer” would set its Variable "OrderReceived" to "true" using the
2011 record element

2012 • The exchange “badPurchaseOrderAckException” specifies that an
2013 Exception of “badPOAckType” Exception Type could occur at both parties

2014

```
2015 <?xml version="1.0" encoding="UTF-8"?>
2016 <package xmlns="http://www.w3.org/2004/12/ws-chor/cdl"
2017      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
2018      xmlns:xsd="http://www.w3.org/2001/XMLSchema"
2019      targetNamespace="http://www.oracle.com/ashwini/sample"
```

```
      name="ConsumerRetailerChoreography"
      version="1.0">

   <informationType name="purchaseOrderType" type="tns:PurchaseOrderMsg"/>
   <informationType name="purchaseOrderAckType" type="tns:PurchaseOrderAckMsg"/>
   <informationType name="badPOAckType"  type="xsd:string" exceptionType="true"/>

   <token name="purchaseOrderID" informationType="tns:intType"/>
   <token name="retailerRef" informationType="tns:uriType"/>
   <tokenLocator tokenName="tns:purchaseOrderID"
                 informationType="tns:purchaseOrderType" query="/PO/orderId"/>
   <tokenLocator tokenName="tns:purchaseOrderID"
                 informationType="tns:purchaseOrderAckType" query="/PO/orderId"/>
   <roleType name="Consumer">
     <behavior name="consumerForRetailer" interface="tns:ConsumerRetailerPT"/>
     <behavior name="consumerForWarehouse" interface="tns:ConsumerWarehousePT"/>
   </roleType>
   <roleType name="Retailer">
     <behavior name="retailerForConsumer" interface="tns:RetailerConsumerPT"/>
   </roleType>
   <relationshipType name="ConsumerRetailerRelationship">
     <role type="tns:Consumer" behavior="consumerForRetailer"/>
     <role type="tns:Retailer" behavior="retailerForConsumer"/>
   </relationshipType>
   <channelType name="ConsumerChannel">
     <role type="tns:Consumer"/>
     <reference>
       <token name="tns:consumerRef"/>
     </reference>
     <identity>
       <token name="tns:purchaseOrderID"/>
     </identity>
   </channelType>
   <channelType name="RetailerChannel">
     <passing channel="ConsumerChannel" action="request" />
     <role type="tns:Retailer" behavior="retailerForConsumer"/>
     <reference>
       <token name="tns:retailerRef"/>
     </reference>
     <identity>
       <token name="tns:purchaseOrderID"/>
     </identity>
   </channelType>

   <choreography name="ConsumerRetailerChoreography" root="true">
     <relationship type="tns:ConsumerRetailerRelationship"/>
     <variableDefinitions>
       <variable name="purchaseOrder" informationType="tns:purchaseOrderType"
                 silent="true" />
       <variable name="purchaseOrderAck"
                 informationType="tns:purchaseOrderAckType" />
       <variable name="retailer-channel" channelType="tns:RetailerChannel"/>
       <variable name="consumer-channel" channelType="tns:ConsumerChannel"/>
       <variable name="badPurchaseOrderAck"
                 informationType="tns:badPOAckType" roleTypes="tns:Consumer"/>
       <variable name="badPurchaseOrderAck"
                 informationType="tns:badPOAckType" roleTypes="tns:Retailer"
                 silent="true" />
     </variableDefinitions>

     <interaction name="createPO"
                  channelVariable="tns:retailer-channel"
```

```
                  initiate="true">
       <participate relationshipType="tns:ConsumerRetailerRelationship"
                    fromRole="tns:Consumer" toRole="tns:Retailer"/>
       <exchange name="request"
                 informationType="tns:purchaseOrderType" action="request">
         <send variable="cdl:getVariable(“tns:purchaseOrder”, “”, “”)" />
         <receive variable="cdl:getVariable(“tns:purchaseOrder”, “”, “”)"
                  recordReference="record-the-channel-info" />
       </exchange>
       <exchange name="response"
                 informationType="purchaseOrderAckType" action="respond">
         <send variable="cdl:getVariable(“tns:purchaseOrderAck”, “”, “”)" />
         <receive variable="cdl:getVariable(“tns:purchaseOrderAck”, “”, “”)" />
       </exchange>
       <exchange name="badPurchaseOrderAckException"
                 informationType="badPOAckType" action="respond">
         <send variable="cdl:getVariable('tns:badPurchaseOrderAck', '', '')"
                 causeException="true" />
         <receive variable="cdl:getVariable(“tns:badPurchaseOrderAck”, “”, “”)"
                 causeException="true" />
       </exchange>
       <record name="record-the-channel-info" when="after">
         <source variable="cdl:getVariable(“tns:purchaseOrder, “”,
                                           “PO/CustomerRef”)"/>
         <target variable="cdl:getVariable(“tns:consumer-channel”, “”, “”)"/>
       </record>
     </interaction>
  </choreography>
</package>
```

2.5.3 Composing Choreographies

The *perform* activity realizes the “composition of Choreographies”, whereas combining existing Choreographies results in the creation of new Choreographies. For example if two separate Choreographies were defined as follows:

- A “Request for Quote” (“RFQ”) Choreography that involves a “Buyer” Role sending a request for a quotation for goods and services to a “Supplier” Role to which the “Supplier” Role responds with either a "Quotation" or a "Decline to Quote" message, and
- An “Order Placement” Choreography, where the “Buyer” Role places and order for goods or services and the “Supplier” Role either accepts the order or rejects it

One could then create a new "Quote and Order" Choreography by reusing the two, where the “RFQ” Choreography was performed first, and then, depending on the outcome of the “RFQ” Choreography, the order is placed using the “Order Placement” Choreography. In this case the new Choreography is "composed" out of the two previously defined Choreographies. Using this approach, Choreographies can be combined to support Choreographies of any required complexity, allowing more flexibility as Choreographies defined elsewhere can be reused.

2133 The perform activity enables a Choreography to specify that another
2134 Choreography is performed at this point in its definition, as an enclosed
2135 Choreography. The performed Choreography, even when defined in a different
2136 Choreography Package, is conceptually treated as an enclosed Choreography.

2137

2138 The syntax of the *perform* construct is:

2139

```
2140 <perform  choreographyName="qname"
2141           choreographyInstanceId="XPath-expression"? >
2142    <bind  name="ncname">
2143      <this variable="XPath-expression" role="qname"/>
2144      <free variable="XPath-expression" role="qname"/>
2145    </bind>*
2146    Choreography-Notation?
2147 </perform>
```

2148

2149 Within the perform element, the choreographyName attribute references the name of
2150 the Choreography to be performed.

2151 The OPTIONAL choreographyInstanceId attribute defines an identifier for this
2152 performance of the Choreography identified by the choreographyName attribute. If
2153 the performed Choreography can only be performed once within the enclosing
2154 Choreography, the choreographyInstanceId attribute is OPTIONAL. Otherwise it
2155 MUST be specified and the value MUST be different for each performance. This
2156 is a dynamic requirement. For example, if a single perform element appears in a
2157 Work Unit that can repeat, each use of perform must assign a different
2158 ChoreographyInstanceId identifier.

2159 The OPTIONAL Choreography-Notation within the perform element defines a
2160 Locally defined Choreography that is performed only by this perform activity. If
2161 specified, the choreographyName attribute within the perform element MUST match
2162 the attribute name within the choreography element of the Choreography-Notation.

2163 The OPTIONAL bind element within the perform element enables information in
2164 the performing Choreography to be shared with the performed Choreography
2165 and vice versa. Within the bind element, the attribute name is used for specifying a
2166 name for each bind element declared within this perform activity. Within the bind
2167 element, the role attribute aliases the Roles from the performing Choreography to
2168 the performed Choreography.

2169 The variable attribute within this element specifies that a Variable in the performing
2170 Choreography is bound with the Variable identified by the variable attribute within
2171 the free element in the performed Choreography.

2172 The following rules apply:

2173 • The Choreography to be performed MUST be either a Locally defined
2174 Choreography that is immediately contained within the performing
2175 Choreography or a Globally defined Choreography. Performed

Choreographies that are declared in a different Choreography Package MUST be included first before they can be performed

- The Role Types within a single bind element MUST be carried out by the same party, hence they MUST belong to the same Participant Type
- The variable attribute within this element and free element MUST define only the WS-CDL function getVariable
- The free Variables specified within the free element MUST have the attribute free set to "true" in their definition within the performed Choreography
- There MUST not be a cyclic dependency on the Choreographies performed. For example, Choreography “C1” is performing Choreography “C2” which is performing Choreography “C1” again is disallowed

The example below shows a Choreography composition, where a Choreography "PurchaseChoreography” is performing the Globally defined Choreography "RetailerWarehouseChoreography” and aliases the Variable "purchaseOrderAtRetailer" to the Variable "purchaseOrder" defined at the performed Choreography "RetailerWarehouseChoreography”. Once aliased, the Variable “purchaseOrderAtRetailer” extends to the enclosed Choreography and thus these Variables can be used interchangeably for sharing their information.

```
<choreography name="PurchaseChoreography”>
   ...
   <variableDefinitions>
      <variable name="purchaseOrderAtRetailer"
             informationType="purchaseOrder" role="tns:Retailer"/>
   </variableDefinitions>
   ...
   <perform choreographyName="RetailerWarehouseChoreography”>
      <bind name="aliasRetailer">
        <this variable="cdl:getVariable(“tns:purchaseOrderAtRetailer”, “”, “”)"
             role="tns:Retailer"/>
        <free variable="cdl:getVariable(“tns:purchaseOrder”, “”, “”)"
             role="tns:Retailer"/>
      </bind>
   </perform>
    ...
</choreography>

<choreography name="RetailerWarehouseChoreography”>
    <variableDefinitions>
       <variable name="purchaseOrder"
          informationType="purchaseOrder" role="tns:Retailer" free=”true”/>
    </variableDefinitions>
     ...
</choreography>
```

2.5.4 Assigning Variables

The *Assign* activity is used to create or change, and then make available within one Role, the value of one or more Variables using the value of another Variable or expression.

The assign activity MAY also be used to cause an Exception at a Role.

The syntax of the *assign* construct is:

```
<assign  roleType="qname">
   <copy  name="ncname" causeException="true"|"false"? >
      <source  variable="XPath-expression"?|expression="Xpath-expression"? />
      <target  variable="XPath-expression" />
   </copy>+
</assign>
```

The copy element within the assign element creates or changes, at the Role specified by the roleType attribute, the Variable defined by the target element using the Variable or expression defined by the source element at the same Role. Within the copy element, the attribute name is used for specifying a name for each copy element declared within this assign activity.

The following rules apply to assignment:

- The source MUST define either a variable attribute or an expression attribute:
  - When the source defines an expression attribute, it MUST contain expressions, as defined in Section 2.4.3. The resulting type of the defined expression MUST be compatible with the target Variable type
  - When the source defines a Variable, then the source and the target Variable MUST be of compatible type
  - When the source defines a Variable, then the source and the target Variable MUST be defined at the same Role
- When the attribute variable is defined it MUST use only the WS-CDL function getVariable
- The target Variable MUST NOT be defined with the attribute silent set to “true”
- When two or more copy elements belong to the same assign element, then they are performed in the order in which they are defined
- If there are two or more copy elements specified within an assign, then all copy operations MUST complete successfully for the assign to complete successfully. Otherwise, none of the Variables specified in the target attribute will be affected

- The OPTIONAL attribute causeException MAY be set to "true" in a copy element if the target Variable is an Exception Variable. The default for this attribute is "false"
- At most one copy element MAY have the attribute causeException set to "true"
- When the attribute causeException is set to "true" in a copy element, the Role specified by the attribute roleType gets into the Exception state after the assign activity has completed

The examples below show some possible usages of assign.

Example 1:

```
<assign roleType="tns:Retailer">
  <copy name="copyAddressInfo">
    <source variable="cdl:getVariable("PurchaseOrderMsg", “”,
                                      "/PO/CustomerAddress")" />
    <target variable="cdl:getVariable("CustomerAddress", “”, “”)" />
  </copy>
</assign>
```

Example 2:

```
<assign roleType="tns:Retailer">
  <copy name="copyPriceInfo">
    <source expression="(10+237)/34" />
    <target variable=”cdl:getVariable("ProductPrice", "", "", "tns:Retailer")” />
  </copy>
</assign>
```

Example 3:

```
<assign roleType="tns:Customer">
  <copy name="copyLiteral">
    <source expression="Hello World" />
    <target variable=”cdl:getVariable("VarName", "", "", "tns:Customer")” />
  </copy>
</assign>
```

2.5.5 Marking Silent Actions

The *Silent Action* activity is an explicit designator used for marking the point where party specific actions with non-observable operational details MUST be performed. For example, the mechanism for checking the inventory of a warehouse should not be observable to other parties, but the fact that the inventory level does influence the global observable behavior with a buyer party needs to be specified in the Choreography definition.

2309 The syntax of the *silent action* construct is:
2310

```
2311 <silentAction roleType="qname? />
```

2312

2313 The OPTIONAL attribute roleType is used to specify the party at which the silent
2314 action will be performed. If a silent action is defined without a Role Type, it is
2315 implied that the action is performed at all the Role Types that are part of the
2316 Relationships of the Choreography this activity is enclosed within.

2317 2.5.6 Marking the Absence of Actions

2318 The *No Action* activity is an explicit designator used for marking the point where
2319 a party does not perform any action.

2320 The syntax of the *no action* construct is:
2321

```
2322 <noAction roleType="qname? />
```

2323

2324 The OPTIONAL attribute roleType is used to specify the party at which no action
2325 will be performed. If a noAction is defined without a Role Type, it is implied that
2326 no action will be performed at any of the Role Types that are part of the
2327 Relationships of the Choreography this activity is enclosed within.

2328 2.5.7 Finalizing a Choreography

2329 The *finalize* activity is used to enable a specific Finalizer Block in successfully
2330 completed instances of immediately enclosed Choreographies, and thus bring
2331 those Choreographies to defined conclusions.

2332 A Choreography that does not perform any Choreographies that have Finalizer
2333 Block(s) defined MUST NOT have any finalize activities specified within it. A
2334 finalize activity MAY be present within a Choreography that has performed a
2335 Choreography with one or more defined Finalizer Block(s) - that is a finalize
2336 activity can be specified within the Choreography body, within an Exception
2337 Block and within Finalizer Blocks.

2338 For a single performed Choreography instance, at most one of its Finalizer
2339 Block(s) SHOULD be enabled by a finalize activity during the subsequent
2340 progress, including Exception handling and finalization, of the enclosing
2341 Choreography.

2342

2343 The syntax of the *finalize* construct is:

2344

```
2345 <finalize  name="ncname"? >
2346     <finalizerReference
2347             choreographyName="ncname"
2348             choreographyInstanceId="XPath-expression"?
2349             finalizerName="ncname"? />
2350     </finalizerReference>+
2351 </finalize>
```

2352

2353 The OPTIONAL attribute name is used for specifying a distinct name for each
2354 finalize element declared within a Choreography Package.

2355 Each finalizerReference element enables a Finalizer Block in a performed instance
2356 of an immediately enclosed Choreography. Within a finalize element, each
2357 finalizerReference MUST refer to a different performed Choreography instance.

2358 Within the finalizerReference element, the choreographyName attribute identifies the
2359 Choreography referenced by the choreographyName attribute of the perform
2360 construct.

2361 Within the finalizerReference element, the OPTIONAL choreographyInstanceId attribute
2362 identifies the performed Choreography instance to be finalized, using the value
2363 defined by the choreographyInstanceId attribute of the perform construct. The
2364 choreographyInstanceId attribute MAY be omitted if the contract logic of the
2365 performing Choreography is such that only one instance of the Choreography
2366 identified by the choreographyName attribute could have been performed when the
2367 finalize activity is enabled. If more than one instance of the Choreography
2368 identified by the choregraphyName attribute could have been performed, the
2369 choreographyInstanceId attribute MUST be present.

2370 Within the finalizerReference element, the attribute finalizerName indicates which
2371 Finalizer Block is to be enabled in the performed instance. If the targeted,
2372 immediately enclosed, Choreography has only one defined Finalizer Block, then
2373 the finalizerName attribute is OPTIONAL.

2374

2375 In the example below, Choreography “CreditDecider” gets credit authorizations
2376 for two bidders, “A” and “B”, at most one of which can be selected. The
2377 “CreditDecider” performs a “CoordinatedCreditAuthorization” Choreography for
2378 each bidder, and then finalizes each performed Choreography depending on
2379 whether “A”, ”B” or neither was selected.
2380

```
2381 <choreography name="CreditDecider">
2382
2383     <!-- only a snippet is shown here -->
2384
2385     <parallel>
2386         <perform name="creditForA"
2387             choreographyName="CoordinatedCreditAuthorization"
```

```
            <!-- bind such that this does the business for A -->
        </perform>
        <perform name="creditForB"
            choreographyName="CoordinatedCreditAuthorization"
            choreographyInstance="creditForB">
            <!-- bind such that this does the business for A -->
        </perform>
    </parallel>

    <!--  other stuff here -->

    <workunit name="chooseA"
        guard="cdl:getVariable('Chosen',,,'Broker')='A'" >
        <finalize>
            <finalizerReference
                choreographyName="CoordinatedCreditAuthorization"
                choreographyInstanceId="creditForA"
                finalizerName="drawDown"/>
            <finalizerReference
                choreographyName="CoordinatedCreditAuthorization"
                choreographyInstanceId="creditForB"
                finalizerName="replenish"/>
        </finalize>
    </workunit>

    <workunit name="chooseB"
        guard="cdl:getVariable('Chosen',,,'Broker')='B'" >
        <finalize>
            <finalizerReference
                choreographyName="CoordinatedCreditAuthorization"
                choreographyInstanceId="creditForB"
                finalizerName="drawDown"/>
            <finalizerReference
                choreographyName="CoordinatedCreditAuthorization"
                choreographyInstanceId="creditForA"
                finalizerName="replenish"/>
        </finalize>
    </workunit>

    <workunit name="chooseNeither"
        guard="cdl:getVariable('Chosen',,,'Broker')='0'" >
        <finalize>
            <finalizerReference
                choreographyName="CoordinatedCreditAuthorization"
                choreographyInstanceId="creditForA"
                finalizerName="replenish"/>
            <finalizerReference
                choreographyName="CoordinatedCreditAuthorization"
                choreographyInstanceId="creditForB"
                finalizerName="replenish"/>
        </finalize>
    </workunit>
</choreography>
```

3 Example

To be completed

2444 # 4 Relationship with the Security framework

2445 The WS-Security specification [24] provides enhancements to SOAP
2446 messaging to provide quality of protection through message integrity,
2447 message confidentiality, and single message authentication, including a
2448 general-purpose mechanism for associating security tokens with
2449 messages, and a description of how to encode binary security tokens.
2450
2451 As messages can have consequences in the real world, collaboration
2452 parties will impose security requirements on their information
2453 exchanges. WS-Security can be used satisfy many of these requirements.

2454 # 5 Relationship with the Reliable Messaging
2455 framework

2456 The WS-Reliability specification [22] provides a reliable mechanism to exchange
2457 information among collaborating parties. The WS-Reliability specification
2458 prescribes the formats for all information exchanged without placing any
2459 restrictions on the content of the encapsulated business documents. The WS-
2460 Reliability specification supports message exchange patterns, over various
2461 transport protocols (examples are HTTP/S, FTP, SMTP, etc.). The WS-Reliability
2462 specification supports sequencing of messages and guaranteed, exactly once
2463 delivery.

2464 A violation of any of these consistency guarantees results in an “error”, which
2465 MAY be reflected in the Choreography with an Exception.

2466 # 6 Relationship with the Coordination framework

2467 In WS-CDL, *Alignment Interactions* and *Coordinated Choreographies* require
2468 support from a Coordination protocol, where agreement on the outcome among
2469 parties can be reached even in the case of failures and loss of messages. In this
2470 case, the Alignment Interactions and the Coordinated Choreographies MUST be
2471 bound to a Coordination protocol.

2472 # 7 Relationship with the Addressing framework

2473 The WS-Addressing specification [28] provides transport-neutral mechanisms to
2474 address Web services and messages, specifically, XML [9, 10] elements to
2475 identify Web service endpoints and to secure end-to-end endpoint identification
2476 in messages. WS-Addressing enables messaging systems to support message
2477 transmission through networks that include processing nodes such as endpoint

managers, firewalls, and gateways in a transport-neutral manner.

WS-Addressing can be used to convey the reference and correlation information for normalizing expanded Channel Variable information into an uniform format that can be processed independently of transport or application.

The WS-Addressing specification is in progress and the WS-Choreography Working Group will review and comment on developments in this effort on an ongoing basis.

8 Conformance

To be completed

9 Acknowledgments

This document has been produced by the members of the Web Services Choreography Working Group. The chairs of this Working Group are Martin Chapman (Oracle Corporation) and Steve Ross-Talbot (Enigmatec Corporation). The editors would like to thank the Working Group members for their contributions. Members of the Working Group are (at the time of writing):

Abbie Barbir (Nortel Networks), Charlton Barreto (webMethods, Inc.), Carine Bournez (W3C), Gary Brown (Enigmatec Corporation), Anthony Fletcher (Choreology Ltd), Peter Furniss (Choreology Ltd), Jim Hendler (University of Maryland (Mind Lab)), Kohei Honda (Queen Mary and Westerfield College), Nickolas Kavantzas (Oracle Corporation), Yutaka Kudou (Hitachi, Ltd.), Yves Lafon (W3C), Monica Martin (Sun Microsystems, Inc.), Robin Milner (Cambridge University), Jeff Mischkinsky (Oracle Corporation), Bijan Parsia (University of Maryland (Mind Lab)), Greg Ritzinger (Novell), Yoko Seki (Hitachi, Ltd.), Prasad Yendluri (webMethods, Inc.), Nobuko Yoshida (Imperial College London).

Previous members of the Working Group were: Assaf Arkin (Intalio Inc.), Daniel Austin (Sun Microsystems, Inc.), Alistair Barros (DSTC Pty Ltd (CITEC)), Richard Bonneau (IONA), Allen Brown (Microsoft Corporation), Mike Brumbelow (Apple), David Burdett (Commerce One), Ravi Byakod (Intalio Inc.), Michael Champion (Software AG), David Chapell (Sonic Software), Ugo Corda (SeeBeyond Technology Corporation), Fred Cummins (EDS), Jon Dart (TIBCO Software), Jean-Jacques Dubray (Attachmate), William Eidson (TIBCO Software), Colleen Evans (Sonic Software), Keith Evans (Hewlett-Packard), Yaron Goland (BEA Systems), Leonard Greski (W. W. Grainger, Inc.), Ricky Ho (Cisco Systems Inc.), Andre Huertas (Uniform Code Council), Duncan Johnston-Watt (Enigmatec Corporation), Eunju Kim (National Computerization Agency), Mayilraj Krishnan (Cisco Systems Inc.), Melanie Kudela (Uniform Code Council), Bruno Kurtic

2519 (webMethods, Inc.), Paul Lipton (Computer Associates), Kevin Liu (SAP AG),
2520 Francis McCabe (Fujitsu Ltd.), Carol McDonald (Sun Microsystems, Inc.), Greg
2521 Meredith (Microsoft Corporation), Eric Newcomer (IONA), Sanjay Patil (IONA),
2522 Ed Peters (webMethods, Inc.), Steve Pruitt (Novell), Dinesh Shahane (TIBCO
2523 Software), Evren Sirin (University of Maryland (Mind Lab)), Ivana Trickovic (SAP
2524 AG), William Vambenepe (Hewlett-Packard), Jim Webber (Arjuna Technologies
2525 Ltd.), Stuart Wheater (Arjuna Technologies Ltd.), Steven White (SeeBeyond
2526 Technology Corporation), Hadrian Zbarcea (IONA).

2527
10 References

2528 [1] S. Bradner, "Key words for use in RFCs to Indicate Requirement Levels", RFC 2119, Harvard
2529 University, March 1997

2530 [2] T. Berners-Lee, R. Fielding, L. Masinter, "Uniform Resource Identifiers (URI): Generic Syntax",
2531 RFC 2396, MIT/LCS, U.C. Irvine, Xerox Corporation, August 1998.

2532 [3] http://www.w3.org/TR/html401/interaction/forms.html#submit-format

2533 [4] http://www.w3.org/TR/html401/appendix/notes.html#ampersands-in-uris

2534 [5] http://www.w3.org/TR/html401/interaction/forms.html#h-17.13.4

2535 [6] Simple Object Access Protocol (SOAP) 1.1 "http://www.w3.org/TR/2000/NOTE-SOAP-
2536 20000508/"

2537 [7] Web Services Definition Language (WSDL) 2.0

2538 [8] OASIS Committee Draft "Universal Description, Discovery and Integration" version 3.0.2

2539 [9] W3C Recommendation "The XML Specification"

2540 [10] XML-Namespaces " Namespaces in XML, Tim Bray et al., eds., W3C, January 1999"

2541 http://www.w3.org/TR/REC-xml-names

2542 [11] W3C Working Draft "XML Schema Part 1: Structures". This is work in progress.

2543 [12] W3C Working Draft "XML Schema Part 2: Datatypes". This is work in progress.

2544 [13] W3C Recommendation "XML Path Language (XPath) Version 1.0"

2545 [14] "Uniform Resource Identifiers (URI): Generic Syntax," RFC 2396, T. Berners-Lee, R.
2546 Fielding, L. Masinter, MIT/LCS, U.C. Irvine, Xerox Corporation, August 1998.

2547 [15] WSCI: Web Services Choreography Interface 1.0, A. Arkin et.al

2548 [16] XLANG: Web Services for Business Process Design, S. Thatte, 2001 Microsoft Corporation

2549 [17] WSFL: Web Service Flow Language 1.0, F. Leymann, 2001 IBM Corporation

2550 [18] OASIS Working Draft “WS-BPEL: Business Process Execution Language 2.0”. This is work
2551 in progress.

2552 [19] BPMI.org “BPML: Business Process Modeling Language 1.0”

2553 [20] Workflow Management Coalition “XPDL: XML Processing Description Language 1.0”, M.
2554 Marin, R. Norin R. Shapiro

2555 [21] OASIS Working Draft “WS-CAF: Web Services Context, Coordination and Transaction
2556 Framework 1.0”. This is work in progress.

2557 [22] OASIS Working Draft “Web Services Reliability 1.0”. This is work in progress.

2558 [23] The Java Language Specification

2559 [24] OASIS “Web Services Security”

2560 [25] J2EE: Java 2 Platform, Enterprise Edition, Sun Microsystems

2561 [26] ECMA. 2001. Standard ECMA-334: C# Language Specification

2562 [27] “XML Inclusions Version 1.0” http://www.w3.org/TR/xinclude/

2563 [28] Web Services Addressing (WS-Addressing) - W3C Member Submission
2564 10 August 2004

2565 # 11 Last Call Issues

2566 ## 11.1 Issue 1

2567 Due to a lack of clarity in existing XML specifications, the WS-Choreography
2568 Working Group is unable at this time to recommend an approach for accessing
2569 and modifying members of lists and arrays.

2570 ## 11.2 Issue 2

2571 The WS-Choreography Working Group is working on a proposal for extending
2572 Choreographies (that is specifying a choreography by defining how it is based on
2573 another choreography). This work is not finalized as of yet, but we do not believe
2574 it will have a major impact on the architecture.

12 WS-CDL XSD Schemas

```
<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
    xmlns:cdl="http://www.w3.org/2004/12/ws-chor/cdl"
    targetNamespace="http://www.w3.org/2004/12/ws-chor/cdl"
    elementFormDefault="qualified">

  <complexType name="tExtensibleElements">
    <annotation>
      <documentation>
        This type is extended by other CDL component types to allow
          elements and attributes from other namespaces to be added.
        This type also contains the optional description element that
        is applied to all CDL constructs.
       </documentation>
    </annotation>
    <sequence>
      <element name="description" minOccurs="0">
        <complexType mixed="true">
          <sequence minOccurs="0" maxOccurs="unbounded">
            <any processContents="lax"/>
          </sequence>
          <attribute name="type" type="cdl:tDescriptionType" use="optional"
                 default="documentation"/>
        </complexType>
      </element>
      <any namespace="##other" processContents="lax"
          minOccurs="0" maxOccurs="unbounded"/>
    </sequence>
    <anyAttribute namespace="##other" processContents="lax"/>
  </complexType>

    <element name="package" type="cdl:tPackage"/>

  <complexType name="tPackage">
    <complexContent>
      <extension base="cdl:tExtensibleElements">
        <sequence>
          <element name="informationType" type="cdl:tInformationType"
                  minOccurs="0" maxOccurs="unbounded"/>
          <element name="token" type="cdl:tToken" minOccurs="0"
                  maxOccurs="unbounded"/>
          <element name="tokenLocator" type="cdl:tTokenLocator"
                  minOccurs="0" maxOccurs="unbounded"/>
          <element name="roleType" type="cdl:tRoleType" minOccurs="0"
                  maxOccurs="unbounded"/>
          <element name="relationshipType" type="cdl:tRelationshipType"
                  minOccurs="0" maxOccurs="unbounded"/>
          <element name="participantType" type="cdl:tParticipantType"
                  minOccurs="0" maxOccurs="unbounded"/>
          <element name="channelType" type="cdl:tChannelType"
                  minOccurs="0" maxOccurs="unbounded"/>
          <element name="choreography" type="cdl:tChoreography"
                  minOccurs="0" maxOccurs="unbounded"/>
        </sequence>
        <attribute name="name" type="NCName" use="required"/>
        <attribute name="author" type="string" use="optional"/>
        <attribute name="version" type="string" use="optional"/>
        <attribute name="targetNamespace" type="anyURI"
                 use="required"/>
      </extension>
```

```
2636      </complexContent>
2637    </complexType>
2638
2639    <complexType name="tInformationType">
2640      <complexContent>
2641        <extension base="cdl:tExtensibleElements">
2642          <attribute name="name" type="NCName" use="required"/>
2643          <attribute name="type" type="QName" use="optional"/>
2644          <attribute name="element" type="QName" use="optional"/>
2645          <attribute name="exceptionType" type="boolean" use="optional"
2646                     default="false" />
2647        </extension>
2648      </complexContent>
2649    </complexType>
2650
2651    <complexType name="tToken">
2652      <complexContent>
2653        <extension base="cdl:tExtensibleElements">
2654          <attribute name="name" type="NCName" use="required"/>
2655          <attribute name="informationType" type="QName"
2656                   use="required"/>
2657        </extension>
2658      </complexContent>
2659    </complexType>
2660
2661    <complexType name="tTokenLocator">
2662      <complexContent>
2663        <extension base="cdl:tExtensibleElements">
2664          <attribute name="tokenName" type="QName" use="required"/>
2665          <attribute name="informationType" type="QName"
2666                   use="required"/>
2667          <attribute name="part" type="NCName" use="optional" />
2668          <attribute name="query" type="cdl:tXPath-expr"
2669                   use="required"/>
2670        </extension>
2671      </complexContent>
2672    </complexType>
2673
2674    <complexType name="tRoleType">
2675      <complexContent>
2676        <extension base="cdl:tExtensibleElements">
2677          <sequence>
2678            <element name="behavior" type="cdl:tBehavior"
2679                    maxOccurs="unbounded"/>
2680          </sequence>
2681          <attribute name="name" type="NCName" use="required"/>
2682        </extension>
2683      </complexContent>
2684    </complexType>
2685
2686    <complexType name="tBehavior">
2687      <complexContent>
2688        <extension base="cdl:tExtensibleElements">
2689          <attribute name="name" type="NCName" use="required"/>
2690          <attribute name="interface" type="QName" use="optional"/>
2691        </extension>
2692      </complexContent>
2693    </complexType>
2694
2695    <complexType name="tRelationshipType">
2696      <complexContent>
2697        <extension base="cdl:tExtensibleElements">
2698          <sequence>
```

```
2699            <element name="role" type="cdl:tRoleRef" minOccurs="2"
2700                    maxOccurs="2"/>
2701          </sequence>
2702          <attribute name="name" type="NCName" use="required"/>
2703        </extension>
2704      </complexContent>
2705    </complexType>
2706
2707    <complexType name="tRoleRef">
2708      <complexContent>
2709        <extension base="cdl:tExtensibleElements">
2710          <attribute name="type" type="QName" use="required"/>
2711          <attribute name="behavior" use="optional">
2712            <simpleType>
2713               <list itemType="NCName"/>
2714            </simpleType>
2715          </attribute>
2716        </extension>
2717      </complexContent>
2718    </complexType>
2719
2720    <complexType name="tParticipantType">
2721      <complexContent>
2722        <extension base="cdl:tExtensibleElements">
2723          <sequence>
2724            <element name="role" type="cdl:tRoleRef2"
2725                    maxOccurs="unbounded"/>
2726          </sequence>
2727          <attribute name="name" type="NCName" use="required"/>
2728        </extension>
2729      </complexContent>
2730    </complexType>
2731
2732    <complexType name="tRoleRef2">
2733      <complexContent>
2734        <extension base="cdl:tExtensibleElements">
2735          <attribute name="type" type="QName" use="required"/>
2736        </extension>
2737      </complexContent>
2738    </complexType>
2739
2740    <complexType name="tChannelType">
2741      <complexContent>
2742        <extension base="cdl:tExtensibleElements">
2743          <sequence>
2744            <element name="passing" type="cdl:tPassing" minOccurs="0"
2745                    maxOccurs="unbounded"/>
2746            <element name="role" type="cdl:tRoleRef3"/>
2747            <element name="reference" type="cdl:tReference"/>
2748            <element name="identity" type="cdl:tIdentity" minOccurs="0"
2749                    maxOccurs="1"/>
2750          </sequence>
2751          <attribute name="name" type="NCName" use="required"/>
2752          <attribute name="usage" type="cdl:tUsage" use="optional"
2753                       default="unlimited"/>
2754          <attribute name="action" type="cdl:tAction" use="optional"
2755                       default="request"/>
2756        </extension>
2757      </complexContent>
2758    </complexType>
2759
2760    <complexType name="tRoleRef3">
2761      <complexContent>
```

```
2762        <extension base="cdl:tExtensibleElements">
2763          <attribute name="type" type="QName" use="required"/>
2764          <attribute name="behavior" type="NCName" use="optional"/>
2765        </extension>
2766      </complexContent>
2767    </complexType>
2768
2769    <complexType name="tPassing">
2770      <complexContent>
2771        <extension base="cdl:tExtensibleElements">
2772          <attribute name="channel" type="QName" use="required"/>
2773          <attribute name="action" type="cdl:tAction" use="optional"
2774                   default="request"/>
2775          <attribute name="new" type="boolean" use="optional"
2776                   default="false"/>
2777        </extension>
2778      </complexContent>
2779    </complexType>
2780
2781    <complexType name="tReference">
2782      <complexContent>
2783        <extension base="cdl:tExtensibleElements">
2784          <sequence>
2785            <element name="token" type="cdl:tTokenReference"
2786                         minOccurs="1" maxOccurs="1"/>
2787          </sequence>
2788        </extension>
2789      </complexContent>
2790    </complexType>
2791
2792    <complexType name="tTokenReference">
2793      <complexContent>
2794        <extension base="cdl:tExtensibleElements">
2795          <attribute name="name" type="QName" use="required"/>
2796        </extension>
2797      </complexContent>
2798    </complexType>
2799
2800    <complexType name="tIdentity">
2801      <complexContent>
2802        <extension base="cdl:tExtensibleElements">
2803          <sequence>
2804            <element name="token" type="cdl:tTokenReference"
2805                    minOccurs="1" maxOccurs="unbounded"/>
2806          </sequence>
2807        </extension>
2808      </complexContent>
2809    </complexType>
2810
2811
2812    <complexType name="tChoreography">
2813      <complexContent>
2814        <extension base="cdl:tExtensibleElements">
2815          <sequence>
2816            <element name="relationship" type="cdl:tRelationshipRef"
2817                    maxOccurs="unbounded"/>
2818            <element name="variableDefinitions"
2819                    type="cdl:tVariableDefinitions" minOccurs="0"/>
2820            <element name="choreography" type="cdl:tChoreography"
2821                     minOccurs="0" maxOccurs="unbounded"/>
2822            <group ref="cdl:activity"/>
2823            <element name="exceptionBlock" type="cdl:tException"
2824                    minOccurs="0"/>
```

```
2825            <element name="finalizerBlock" type="cdl:tFinalizer"
2826                         minOccurs="0" maxOccurs="unbounded"/>
2827          </sequence>
2828          <attribute name="name" type="NCName" use="required"/>
2829          <attribute name="complete" type="cdl:tBoolean-expr"
2830                      use="optional"/>
2831          <attribute name="isolation" type="boolean"
2832                      use="optional" default="false"/>
2833          <attribute name="root" type="boolean" use="optional"
2834                      default="false"/>
2835          <attribute name="coordination" type="boolean" use="optional"
2836                      default="false"/>
2837        </extension>
2838      </complexContent>
2839    </complexType>
2840
2841    <complexType name="tRelationshipRef">
2842      <complexContent>
2843        <extension base="cdl:tExtensibleElements">
2844          <attribute name="type" type="QName" use="required"/>
2845        </extension>
2846      </complexContent>
2847    </complexType>
2848
2849    <complexType name="tVariableDefinitions">
2850      <complexContent>
2851        <extension base="cdl:tExtensibleElements">
2852          <sequence>
2853            <element name="variable" type="cdl:tVariable"
2854                    maxOccurs="unbounded"/>
2855          </sequence>
2856        </extension>
2857      </complexContent>
2858    </complexType>
2859
2860    <complexType name="tVariable">
2861      <complexContent>
2862        <extension base="cdl:tExtensibleElements">
2863          <attribute name="name" type="NCName" use="required"/>
2864          <attribute name="informationType" type="QName"
2865                  use="optional"/>
2866          <attribute name="channelType" type="QName" use="optional"/>
2867          <attribute name="mutable" type="boolean" use="optional"
2868                  default="true"/>
2869          <attribute name="free" type="boolean" use="optional"
2870                  default="false"/>
2871          <attribute name="silent" type="boolean" use="optional"
2872                  default="false"/>
2873          <attribute name="roleTypes" use="optional">
2874             <simpleType>
2875                <list itemType="QName"/>
2876             </simpleType>
2877          </attribute>
2878        </extension>
2879      </complexContent>
2880    </complexType>
2881
2882    <group name="activity">
2883      <choice>
2884        <element name="sequence" type="cdl:tSequence"/>
2885        <element name="parallel" type="cdl:tParallel"/>
2886        <element name="choice" type="cdl:tChoice"/>
2887        <element name="workunit" type="cdl:tWorkunit"/>
```

```
2888        <element name="interaction" type="cdl:tInteraction"/>
2889        <element name="perform" type="cdl:tPerform"/>
2890        <element name="assign" type="cdl:tAssign"/>
2891        <element name="silentAction" type="cdl:tSilentAction"/>
2892        <element name="noAction" type="cdl:tNoAction"/>
2893        <element name="finalize" type="cdl:tFinalize"/>
2894
2895      </choice>
2896    </group>
2897
2898    <complexType name="tSequence">
2899      <complexContent>
2900        <extension base="cdl:tExtensibleElements">
2901          <sequence>
2902            <group ref="cdl:activity" maxOccurs="unbounded"/>
2903          </sequence>
2904        </extension>
2905      </complexContent>
2906    </complexType>
2907
2908    <complexType name="tParallel">
2909      <complexContent>
2910        <extension base="cdl:tExtensibleElements">
2911          <sequence>
2912            <group ref="cdl:activity" maxOccurs="unbounded"/>
2913          </sequence>
2914        </extension>
2915      </complexContent>
2916    </complexType>
2917    <complexType name="tChoice">
2918      <complexContent>
2919        <extension base="cdl:tExtensibleElements">
2920          <sequence>
2921            <group ref="cdl:activity" maxOccurs="unbounded"/>
2922          </sequence>
2923        </extension>
2924      </complexContent>
2925    </complexType>
2926
2927    <complexType name="tWorkunit">
2928      <complexContent>
2929        <extension base="cdl:tExtensibleElements">
2930          <sequence>
2931            <group ref="cdl:activity"/>
2932          </sequence>
2933          <attribute name="name" type="NCName" use="required"/>
2934          <attribute name="guard" type="cdl:tBoolean-expr"
2935                   use="optional"/>
2936          <attribute name="repeat" type="cdl:tBoolean-expr"
2937                   use="optional"/>
2938          <attribute name="block" type="boolean"
2939                   use="optional" default="false"/>
2940        </extension>
2941      </complexContent>
2942    </complexType>
2943
2944    <complexType name="tPerform">
2945      <complexContent>
2946        <extension base="cdl:tExtensibleElements">
2947          <sequence>
2948            <element name="bind" type="cdl:tBind"
2949                    minOccurs="0" maxOccurs="unbounded"/>
2950            <element name="choreography" type="cdl:tChoreography"
```

```
2951                    minOccurs="0" maxOccurs="1"/>
2952          </sequence>
2953          <attribute name="choreographyName" type="QName" use="required"/>
2954          <attribute name="choreographyInstanceId" type="cdl:tXPath-expr"
2955  use="optional"/>
2956        </extension>
2957      </complexContent>
2958    </complexType>
2959
2960    <complexType name="tBind">
2961      <complexContent>
2962        <extension base="cdl:tExtensibleElements">
2963          <sequence>
2964            <element name="this" type="cdl:tBindVariable"/>
2965            <element name="free" type="cdl:tBindVariable"/>
2966          </sequence>
2967           <attribute name="name" type="NCName" use="required"/>
2968        </extension>
2969      </complexContent>
2970    </complexType>
2971
2972    <complexType name="tBindVariable">
2973      <complexContent>
2974        <extension base="cdl:tExtensibleElements">
2975          <attribute name="variable" type="cdl:tXPath-expr"
2976                   use="required"/>
2977          <attribute name="role" type="QName" use="required"/>
2978        </extension>
2979      </complexContent>
2980    </complexType>
2981
2982    <complexType name="tInteraction">
2983      <complexContent>
2984        <extension base="cdl:tExtensibleElements">
2985          <sequence>
2986            <element name="participate" type="cdl:tParticipate"/>
2987            <element name="exchange" type="cdl:tExchange" minOccurs="0"
2988                    maxOccurs="unbounded"/>
2989            <element name="timeout" type="cdl:tTimeout" minOccurs="0"
2990                    maxOccurs="1"/>
2991            <element name="record" type="cdl:tRecord" minOccurs="0"
2992                    maxOccurs="unbounded"/>
2993          </sequence>
2994          <attribute name="name" type="NCName" use="required"/>
2995          <attribute name="channelVariable" type="QName"
2996                   use="required"/>
2997          <attribute name="operation" type="NCName" use="required"/>
2998          <attribute name="align" type="boolean" use="optional"
2999                   default="false"/>
3000          <attribute name="initiate" type="boolean"
3001                   use="optional" default="false"/>
3002        </extension>
3003      </complexContent>
3004    </complexType>
3005
3006    <complexType name="tTimeout">
3007      <complexContent>
3008        <extension base="cdl:tExtensibleElements">
3009          <attribute name="time-to-complete" type="cdl:tXPath-expr" use="required"/>
3010          <attribute name="fromRoleRecordReference" use="optional">
3011            <simpleType>
3012               <list itemType="NCName"/>
3013            </simpleType>
```

```
3014          </attribute>
3015          <attribute name="toRoleRecordReference" use="optional">
3016            <simpleType>
3017               <list itemType="NCName"/>
3018            </simpleType>
3019          </attribute>
3020        </extension>
3021      </complexContent>
3022    </complexType>
3023
3024    <complexType name="tParticipate">
3025      <complexContent>
3026        <extension base="cdl:tExtensibleElements">
3027          <attribute name="relationshipType" type="QName" use="required"/>
3028          <attribute name="fromRole" type="QName" use="required"/>
3029          <attribute name="toRole" type="QName" use="required"/>
3030        </extension>
3031      </complexContent>
3032    </complexType>
3033
3034    <complexType name="tExchange">
3035      <complexContent>
3036        <extension base="cdl:tExtensibleElements">
3037          <sequence>
3038            <element name="send" type="cdl:tVariableRecordRef"/>
3039            <element name="receive" type="cdl:tVariableRecordRef"/>
3040          </sequence>
3041          <attribute name="name" type="NCName" use="required"/>
3042          <attribute name="informationType" type="QName"
3043                   use="optional"/>
3044          <attribute name="channelType" type="QName"
3045                   use="optional"/>
3046          <attribute name="action" type="cdl:tAction2" use="required"/>
3047        </extension>
3048      </complexContent>
3049    </complexType>
3050
3051    <complexType name="tVariableRecordRef">
3052      <complexContent>
3053        <extension base="cdl:tExtensibleElements">
3054          <attribute name="variable" type="cdl:tXPath-expr"
3055                  use="optional"/>
3056          <attribute name="recordReference" use="optional">
3057            <simpleType>
3058               <list itemType="NCName"/>
3059            </simpleType>
3060          </attribute>
3061          <attribute name="causeException" type="boolean"
3062                  use="optional" default="false"/>
3063        </extension>
3064      </complexContent>
3065    </complexType>
3066
3067    <complexType name="tRecord">
3068      <complexContent>
3069        <extension base="cdl:tExtensibleElements">
3070          <sequence>
3071            <element name="source" type="cdl:tSourceVariableRef"/>
3072            <element name="target" type="cdl:tVariableRef"/>
3073          </sequence>
3074          <attribute name="name" type="NCName" use="required"/>
3075          <attribute name="causeException" type="boolean" use="optional"
3076  default="false"/>
```

```
3077          <attribute name="when" type="cdl:tWhenType" use="required"/>
3078        </extension>
3079      </complexContent>
3080    </complexType>
3081
3082     <complexType name="tSourceVariableRef">
3083      <complexContent>
3084        <extension base="cdl:tExtensibleElements">
3085          <attribute name="variable" type="cdl:tXPath-expr"
3086                  use="optional"/>
3087          <attribute name="expression" type="cdl:tXPath-expr"
3088                  use="optional"/>
3089        </extension>
3090      </complexContent>
3091    </complexType>
3092
3093    <complexType name="tVariableRef">
3094      <complexContent>
3095        <extension base="cdl:tExtensibleElements">
3096          <attribute name="variable" type="cdl:tXPath-expr"
3097                  use="required"/>
3098        </extension>
3099      </complexContent>
3100    </complexType>
3101
3102    <complexType name="tAssign">
3103      <complexContent>
3104        <extension base="cdl:tExtensibleElements">
3105          <sequence>
3106        <element name="copy" type="cdl:tCopy"
3107                 maxOccurs="unbounded"/>
3108          </sequence>
3109          <attribute name="roleType" type="QName" use="required"/>
3110        </extension>
3111      </complexContent>
3112    </complexType>
3113
3114    <complexType name="tCopy">
3115      <complexContent>
3116        <extension base="cdl:tExtensibleElements">
3117          <sequence>
3118            <element name="source" type="cdl:tSourceVariableRef"/>
3119            <element name="target" type="cdl:tVariableRef"/>
3120          </sequence>
3121          <attribute name="name" type="NCName" use="required"/>
3122          <attribute name="causeException" type="boolean"
3123                  use="optional" default="false"/>
3124        </extension>
3125      </complexContent>
3126    </complexType>
3127
3128    <complexType name="tSilentAction">
3129      <complexContent>
3130        <extension base="cdl:tExtensibleElements">
3131          <attribute name="roleType" type="QName" use="optional"/>
3132        </extension>
3133      </complexContent>
3134    </complexType>
3135
3136    <complexType name="tNoAction">
3137      <complexContent>
3138        <extension base="cdl:tExtensibleElements">
3139          <attribute name="roleType" type="QName" use="optional"/>
```

```
3140          </extension>
3141        </complexContent>
3142      </complexType>
3143
3144      <complexType name="tFinalize">
3145        <complexContent>
3146          <extension base="cdl:tExtensibleElements">
3147            <sequence>
3148              <element name="finalizerReference" type="cdl:tFinalizerReference"
3149                                       maxOccurs="unbounded"/>
3150            </sequence>
3151            <attribute name="name" type="NCName" use="required"/>
3152          </extension>
3153        </complexContent>
3154      </complexType>
3155
3156      <complexType name="tFinalizerReference">
3157        <complexContent>
3158          <extension base="cdl:tExtensibleElements">
3159            <attribute name="choreographyName" type="NCName" use="required"/>
3160            <attribute name="choreographyInstanceId" type="cdl:tXPath-expr"
3161                                                            use="optional"/>
3162            <attribute name="finalizerName" type="NCName" use="optional"/>
3163          </extension>
3164        </complexContent>
3165      </complexType>
3166
3167      <complexType name="tException">
3168        <complexContent>
3169          <extension base="cdl:tExtensibleElements">
3170            <sequence>
3171              <element name="workunit" type="cdl:tWorkunit"
3172                       maxOccurs="unbounded"/>
3173            </sequence>
3174            <attribute name="name" type="NCName" use="required"/>
3175          </extension>
3176        </complexContent>
3177      </complexType>
3178
3179      <complexType name="tFinalizer">
3180        <complexContent>
3181          <extension base="cdl:tExtensibleElements">
3182            <sequence>
3183              <element name="workunit" type="cdl:tWorkunit"/>
3184            </sequence>
3185            <attribute name="name" type="NCName" use="required"/>
3186          </extension>
3187        </complexContent>
3188      </complexType>
3189
3190      <simpleType name="tAction">
3191        <restriction base="string">
3192          <enumeration value="request-respond"/>
3193          <enumeration value="request"/>
3194          <enumeration value="respond"/>
3195        </restriction>
3196      </simpleType>
3197
3198      <simpleType name="tAction2">
3199        <restriction base="string">
3200          <enumeration value="request"/>
3201          <enumeration value="respond"/>
3202        </restriction>
```

```
3203      </simpleType>
3204
3205      <simpleType name="tUsage">
3206        <restriction base="string">
3207          <enumeration value="once"/>
3208          <enumeration value="unlimited"/>
3209        </restriction>
3210      </simpleType>
3211
3212      <simpleType name="tWhenType">
3213        <restriction base="string">
3214          <enumeration value="before"/>
3215          <enumeration value="after"/>
3216          <enumeration value="timeout"/>
3217        </restriction>
3218      </simpleType>
3219
3220
3221      <simpleType name="tBoolean-expr">
3222        <restriction base="string"/>
3223      </simpleType>
3224
3225      <simpleType name="tXPath-expr">
3226        <restriction base="string"/>
3227      </simpleType>
3228
3229      <simpleType name="tDescriptionType">
3230        <restriction base="string">
3231          <enumeration value="documentation"/>
3232          <enumeration value="reference"/>
3233          <enumeration value="semantics"/>
3234        </restriction>
3235      </simpleType>
3236
3237    </schema>
```

3238