



1

2<dt>wd</dt><role>editorsdraft</role><W3CDes>Working Draft</W3CDes>

3<language>us-en</language>

4<t>Web Services Choreography Description

5Language</t>, <v>Version 1.0</v>

6<doct>Editor's Draft</doct>, <d>11</d> <m>November</m>

7<y>2004</y>

8**This version:**

9 TBD

10**Latest version:**

11 TBD

12**Previous Version:**

13 Not Applicable

14**Editors (alphabetically):**

15 <n>Nickolaos Kavantzas</n>, <a>Oracle ,

16 <e>nickolas.kavantzas@oracle.com</e>

17 <n>David Burdett</n>, <a>Commerce One

18 <e>david.burdett@commerceone.com</e>

19 <n>Gregory Ritzinger</n>, <a>Novell <e>gritzinger@novell.com</e>

20Copyright © 2004 ^{W3C®} (MIT, ERCIM, Keio), All Rights Reserved. W3C liability,

21trademark, document use and software licensing rules apply.

22Abstract

23The Web Services Choreography Description Language (WS-CDL) is an XML-
24based language that describes peer-to-peer collaborations of parties by defining,
25from a global viewpoint, their common and complementary observable behavior;
26where ordered message exchanges result in accomplishing a common business
27goal.

28The Web Services specifications offer a communication bridge between the
29heterogeneous computational environments used to develop and host
30applications. The future of E-Business applications requires the ability to perform
31long-lived, peer-to-peer collaborations between the participating services, within
32or across the trusted domains of an organization.

1The Web Services Choreography specification is targeted for composing
2interoperable, peer-to-peer collaborations between any type of party regardless
3of the supporting platform or programming model used by the implementation of
4the hosting environment.

5Status of this Document

6This section describes the status of this document at the time of its publication.
7Other documents may supersede this document. A list of current W3C
8publications and the latest revision of this technical report can be found in the
9[W3C technical reports index](http://www.w3.org/TR/) at <http://www.w3.org/TR/>.

10This is the First Public Working Draft of the Web Services Choreography
11Description Language document.

12It has been produced by the Web Services Choreography Working Group, which
13is part of the Web Services Activity. Although the Working Group agreed to
14request publication of this document, this document does not represent
15consensus within the Working Group about Web Services Choreography
16description language.

17This document is a chartered deliverable of the Web Services Choreography
18Working Group. It is an early stage document and major changes are expected
19in the near future.

20Comments on this document should be sent to [public-ws-chor-
21comments@w3.org](mailto:public-ws-chor-comments@w3.org) (public archive). It is inappropriate to send discussion emails
22to this address.

23Discussion of this document takes place on the public public-ws-chor@w3.org
24mailing list (public archive) per the email communication rules in the Web
25Services Choreography Working Group charter.

26This document has been produced under the 24 January 2002 CPP as amended
27by the W3C Patent Policy Transition Procedure. An individual who has actual
28knowledge of a patent which the individual believes contains Essential Claim(s)
29with respect to this specification should disclose the information in accordance
30with section 6 of the W3C Patent Policy. Patent disclosures relevant to this
31specification may be found on the Working Group's patent disclosure page.

32Publication as a Working Draft does not imply endorsement by the W3C
33Membership. This is a draft document and may be updated, replaced or
34obsoleted by other documents at any time. It is inappropriate to cite this
35document as other than work in progress.

36Revision Description

37This is the second editor's draft of the document.

1 Table of Contents

2	Status of this Document.....	2
3	Revision Description.....	2
4	1 Introduction.....	4
5	1.1 Notational Conventions.....	5
6	1.2 Purpose of the Choreography Language.....	7
7	1.3 Goals.....	9
8	1.4 Relationship with XML and WSDL.....	10
9	1.5 Relationship with Business Process Languages.....	10
10	1.6 Time Assumptions.....	10
11	2 Choreography Model.....	11
12	2.1 Model Overview.....	11
13	2.2 Choreography Document Structure.....	12
14	2.2.1 Package.....	12
15	2.2.2 Choreography document Naming and Linking.....	13
16	2.2.3 Language Extensibility and Binding.....	14
17	2.2.4 Semantics.....	14
18	2.3 Collaborating Parties.....	14
19	2.3.1 Role Types.....	15
20	2.3.2 Relationship Types.....	15
21	2.3.3 Participant Types.....	16
22	2.3.4 Channel Types.....	17
23	2.4 Information Driven Collaborations.....	20
24	2.4.1 Information Types.....	20
25	2.4.2 Variables.....	21
26	2.4.3 Expressions.....	24
27	2.4.3.1 WS-CDL Supplied Functions.....	24
28	2.4.4 Tokens.....	26
29	2.4.5 Choreographies.....	27
30	2.4.6 WorkUnits.....	29
31	2.4.7 Including Choreographies.....	33
32	2.4.8 Choreography Life-line.....	33
33	2.4.9 Choreography Recovery.....	34
34	2.4.9.1 Exception Block.....	34
35	2.4.9.2 Finalizer Block.....	36
36	2.5 Activities.....	36
37	2.5.1 Ordering Structures.....	37
38	2.5.1.1 Sequence.....	37
39	2.5.1.2 Parallel.....	38
40	2.5.1.3 Choice.....	38
41	2.5.2 Interacting.....	39
42	2.5.2.1 Interaction Based Information Alignment.....	39
43	2.5.2.2 Interaction Life-line.....	40
44	2.5.2.3 Interaction Syntax.....	40
45	2.5.3 Composing Choreographies.....	48

1	2.5.4Assigning Variables.....	50
2	2.5.5Marking Silent Actions.....	52
3	2.5.6Marking the Absence of Actions.....	52
43	Example.....	52
54	Relationship with the Security framework.....	52
65	Relationship with the Reliable Messaging framework.....	53
76	Relationship with the Transaction/Coordination framework.....	53
87	Acknowledgments.....	53
98	References.....	53
109	WS-CDL XSD Schemas.....	54

111 Introduction

12For many years, organizations have been developing solutions for automating
13their peer-to-peer collaborations, within or across their trusted domain, in an
14effort to improve productivity and reduce operating costs.

15The past few years have seen the Extensible Markup Language (XML) and the
16Web Services framework developing as the de facto choices for describing
17interoperable data and platform neutral business interfaces, enabling more open
18business transactions to be developed.

19Web Services are a key component of the emerging, loosely coupled, Web-
20based computing architecture. A Web Service is an autonomous, standards-
21based component whose public interfaces are defined and described using XML.
22Other systems may interact with a Web Service in a manner prescribed by its
23definition, using XML based messages conveyed by Internet protocols.

24The Web Services specifications offer a communication bridge between the
25heterogeneous computational environments used to develop and host
26applications. The future of E-Business applications requires the ability to perform
27long-lived, peer-to-peer collaborations between the participating services, within
28or across the trusted domains of an organization.

29The Web Service architecture stack targeted for integrating interacting
30applications consists of the following components:

- 31 • *SOAP*: defines the basic formatting of a message and the basic
32 delivery options independent of programming language, operating
33 system, or platform. A SOAP compliant Web Service knows how to
34 send and receive SOAP-based messages
- 35 • *WSDL*: describes the static interface of a Web Service. It defines the
36 protocol and the message characteristics of end points. Data types are
37 defined by XML Schema specification, which supports rich type
38 definitions and allows expressing any kind of XML type requirement for
39 the application data

- 1 • *Registry*: allows publishing the availability of a Web Service and its
2 discovery from service requesters using sophisticated searching
3 mechanisms
- 4 • *Security layer*: ensures that exchanged information are not modified or
5 forged
- 6 • *Reliable Messaging layer*: provides exactly-once and guaranteed
7 delivery of information exchanged between parties
- 8 • *Context, Coordination and Transaction layer*: defines interoperable
9 mechanisms for propagating context of long-lived business
10 transactions and enables parties to meet correctness requirements by
11 following a global agreement protocol
- 12 • *Business Process Languages layer*: describes the execution logic of
13 Web Services based applications by defining their control flows (such
14 as conditional, sequential, parallel and exceptional execution) and
15 prescribing the rules for consistently managing their non-observable
16 data
- 17 • *Choreography layer*: describes collaborations of parties by defining
18 from a global viewpoint their common and complementary observable
19 behavior, where information exchanges occur, when the jointly agreed
20 ordering rules are satisfied

21The Web Services Choreography specification is targeted for composing
22interoperable, collaborations between any type of party regardless of the
23supporting platform or programming model used by the implementation of the
24hosting environment.

251.1 Notational Conventions

26The keywords "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT",
27"SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in
28this document are to be interpreted as described in RFC-2119 [2].

29The following namespace prefixes are used throughout this document:

Prefix	Namespace URI	Definition
wSDL	http://schemas.xmlsoap.org/wSDL/	WSDL namespace for WSDL framework.
cdl	http://www.w3.org/ws/choreography/2004/09/WSCDL	WSCDL namespace for Choreography 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].

30This specification uses an *informal syntax* to describe the XML grammar of a
31WS-CDL document:

- 32 • The syntax appears as an XML instance, but the values indicate the
33 data types instead of values.
- 34 • Characters are appended to elements and attributes as follows: "?" (0
35 or 1), "*" (0 or more), "+" (1 or more).
- 36 • Elements names ending in "..." (such as <element.../> or <element...>
37 indicate that elements/attributes irrelevant to the context are being
38 omitted.
- 39 • Grammar in bold has not been introduced earlier in the document, or is
40 of particular interest in an example.
- 41 • <-- extensibility element --> is a placeholder for elements from some
42 "other" namespace (like ##other in XSD).
- 43 • The XML namespace prefixes (defined above) are used to indicate the
44 namespace of the element being defined.

- 1 • Examples starting with <?xml contain enough information to conform to
2 this specification; others examples are fragments and require additional
3 information to be specified in order to conform.

4XSD schemas are provided as a formal definition of WS-CDL grammar (see
5Section 9).

61.2 Purpose of the Choreography Language

7Business or other activities that involve multiple different organizations or
8independent processes are engaged in a collaborative fashion to achieve a
9common business goal, such as *Order Fulfillment*.

10For the collaboration to work successfully, the rules of engagement between all
11the interacting parties must be provided. Whereas today these rules are
12frequently written in English, a standardized way for precisely defining these
13interactions, leaving unambiguous documentation of the parties and
14responsibilities of each, is missing.

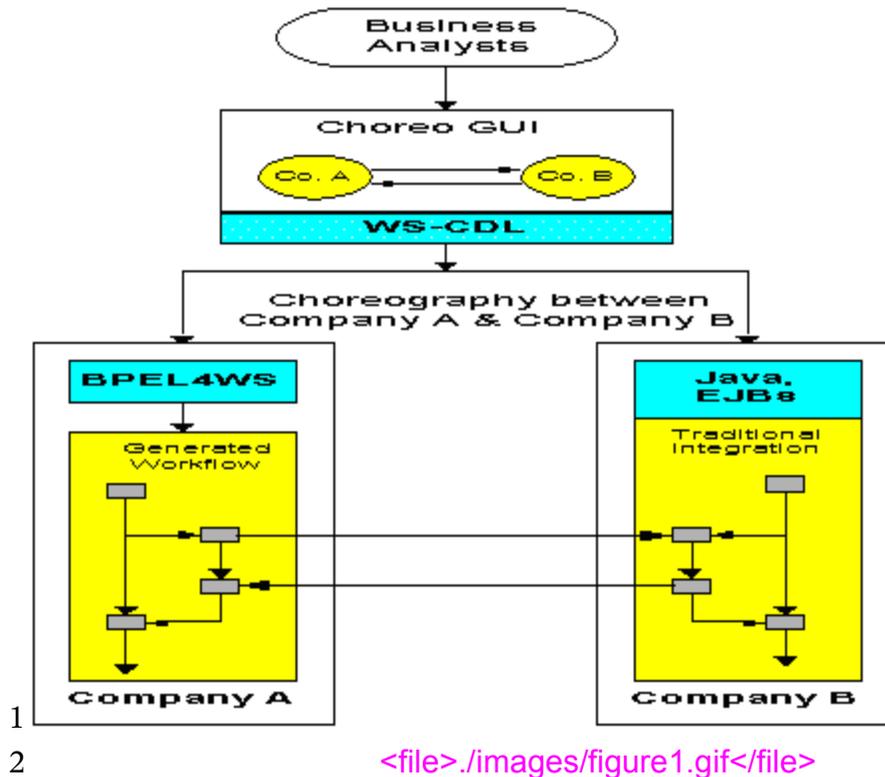
15The Web Services Choreography specification is targeted for precisely
16describing collaborations between any type of party regardless of the supporting
17platform or programming model used by the implementation of the hosting
18environment.

19Using the Web Services Choreography specification, a contract containing a
20"global" definition of the common ordering conditions and constraints under
21which messages are exchanged is produced that describes from a global
22viewpoint the common and complementary observable behavior of all the parties
23involved. Each party can then use the global definition to build and test solutions
24that conform to it.

25The main advantage of a contract with a global definition approach is that it
26separates the process being followed by an individual business or system within
27a "domain of control" from the definition of the sequence in which each business
28or system exchanges information with others. This means that, as long as the
29"observable" sequence does not change, the rules and logic followed within the
30domain of control can change at will.

31In real-world scenarios, corporate entities are often unwilling to delegate control
32of their business processes to their integration partners. Choreography offers a
33means by which the rules of participation within a collaboration can be clearly
34defined and agreed to, jointly. Each entity may then implement its portion of the
35Choreography as determined by the common view.

36The figure below demonstrates a possible usage of the Choreography
37Language.



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

4In Figure 1, Company A and Company B wish to integrate their Web Services
5based applications. The respective business analysts at both companies agree
6upon the services involved in the collaboration, their interactions, and their
7common ordering and constraint rules under which the interactions occur. They
8then generate a Choreography Language based representation. In this example,
9a Choreography specifies the interactions between services across business
10entities ensuring interoperability, while leaving actual implementation decisions
11in the hands of each individual company:

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

18Similarly, a Choreography can specify the interoperability and interactions
19between services within one business entity.

201.3 Goals

21The primary goal of a Choreography Language is to specify a declarative, XML
22based language that defines from a global viewpoint the common and

1complementary observable behavior, where information exchanges occur, and
2when the jointly agreed ordering rules are satisfied.

3Some additional goals of this definition language are to permit:

- 4 • *Reusability*. The same Choreography definition is usable by different
5 parties operating in different contexts (industry, locale, etc.) with
6 different software (e.g. application software)
- 7 • *Cooperation*. Choreographies define the sequence of exchanging
8 messages between two (or more) independent parties or processes by
9 describing how they should cooperate
- 10 • *Multi-Party Collaboration*. Choreographies can be defined involving any
11 number of parties or processes
- 12 • *Semantics*. Choreographies can include human-readable
13 documentation and semantics for all the components in the
14 Choreography
- 15 • *Composability*. Existing Choreographies can be combined to form new
16 Choreographies that may be reused in different contexts
- 17 • *Modularity*. Choreographies can be defined using an "inclusion" facility
18 that allows a Choreography to be created from parts contained in
19 several different Choreographies
- 20 • *Information Driven Collaboration*. Choreographies describe how parties
21 make progress within a collaboration, when recordings of exchanged
22 information and observable information changes cause ordering
23 constraints to be fulfilled
- 24 • *Information Alignment*. Choreographies allow the parties that take part
25 in Choreographies to communicate and synchronize their observable
26 information changes and the actual values of the exchanged
27 information as well
- 28 • *Exception Handling*. Choreographies can define how exceptional or
29 unusual conditions that occur while the Choreography is performed are
30 handled
- 31 • *Transactionality*. The processes or parties that take part in a
32 Choreography can work in a "transactional" way with the ability to
33 coordinate the outcome of the long-lived collaborations, which include
34 multiple, often recursive collaboration units, each with its own business
35 rules and goals
- 36 • *Specification Composability*. This specification will work alongside and
37 complement other specifications such as the WS-Reliability [22], WS-
38 Composite Application Framework (WS-CAF) [21], WS-Security [24],
39 Business Process Execution Language for WS (WS-BPEL) [18], etc.

1 1.4 Relationship with XML and WSDL

2 This specification depends on the following specifications: XML 1.0 [9], XML-
3 Namespaces [10], XML-Schema 1.0 [11, 12] and XPath 1.0 [13]. In addition,
4 support for including and referencing service definitions given in WSDL 2.0 [7] is
5 a normative part of this specification.

6 1.5 Relationship with Business Process Languages

7 A Choreography Language is not an "executable business process description
8 language" or an implementation language. The role of specifying the execution
9 logic of an application will be covered by these specifications [16, 17, 18, 19, 20,
10 23, 26].

11 A Choreography Language does not depend on a specific business process
12 implementation language. Thus, it can be used to specify truly interoperable,
13 collaborations between any type of party regardless of the supporting platform or
14 programming model used by the implementation of the hosting environment.
15 Each party, adhering to a Choreography Language collaboration representation,
16 could be implemented using completely different mechanisms such as:

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

22 1.6 Time Assumptions

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

29 2 Choreography Model

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

32 2.1 Model Overview

33 WS-CDL describes interoperable, collaborations between parties. In order to
34 facilitate these collaborations, services commit to mutual responsibilities by
35 establishing Relationships. Their collaboration takes place in a jointly agreed set

1of ordering and constraint rules, whereby information is exchanged between the
2parties.

3The Choreography model consists of the following notations:

- 4 • *Participant Types, Role Types and Relationship Types* - Within a
5 Choreography, information is always exchanged between parties within
6 the same or across trust boundaries. A Role Type enumerates the
7 observable behavior a party exhibits in order to collaborate with other
8 parties. A Relationship Type identifies the mutual commitments that must
9 be made between two parties for them to collaborate successfully. A
10 Participant Type is grouping together the parts of the observable behavior
11 that must be implemented by the same entity or organization
- 12 • *Types, Variables and Tokens* - Variables contain information about
13 commonly observable objects in a collaboration, such as the information
14 exchanged or the observable information of the Roles involved. Tokens
15 are aliases that can be used to reference parts of a Variable. Both
16 Variables and Tokens have Types that define the structure of what the
17 Variable or Token contains
- 18 • *Choreographies* - Choreographies define collaborations between
19 interacting parties:
 - 20 • *Choreography Life-line* - Choreography Life-line expresses the
21 progression of a collaboration. Initially, the collaboration is started
22 at a specific business process, then work is performed by following
23 the Choreography and finally the Choreography completes, either
24 normally or abnormally
 - 25 • *Choreography Recovery* consists of:
 - 26 • *Choreography Exception Block* - An Exception Block
27 specifies what additional interactions should occur when
28 a Choreography behaves in an abnormal way
 - 29 • *Choreography Finalizer Block* - A Finalizer Block describes
30 what additional interactions should occur to reverse the
31 effect of an earlier successfully completed Choreography
- 32 • *Channels* - A Channel realizes a point of collaboration between parties
33 by specifying where and how information is exchanged
- 34 • *WorkUnits* - A Work Unit prescribes the constraints that must be fulfilled
35 for making progress and thus performing actual work within a
36 Choreography
- 37 • *Activities and Ordering Structures* - Activities are the lowest level
38 components of the Choreography that perform the actual work.
39 Ordering Structures combine activities with other Ordering Structures in
40 a nested structure to express the ordering conditions in which
41 information within the Choreography is exchanged

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

7 2.2 Choreography Document Structure

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

11 2.2.1 Package

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

16

17 The syntax of the *package* construct is:

```
19 <package  
20   name="ncname"  
21   author="xsd:string"?  
22   version="xsd:string"  
23   targetNamespace="uri"  
24   xmlns="http://www.w3.org/ws/choreography/2004/09/WSCDL/">  
25  
26   informationType*  
27   token*  
28   tokenLocator*  
29   roleType*  
30   relationshipType*  
31   participantType*  
32   channelType*  
33  
34   choreography*  
35 </package>
```

36 The Choreography Package contains:

- 37 • Zero or more Information Types
- 38 • Zero or more Tokens and Token Locators
- 39 • Zero or more Role Types
- 40 • Zero or more Relationship Types
- 41 • Zero or more Participant Types

- 1 • Zero or more Channel Types
- 2 • Zero or more Package-level Choreographies

3The top-level attributes name, author, and version define authoring properties of the
4Choreography document.

5The targetNamespace attribute provides the namespace associated with all
6definitions contained in this Package. Choreography definitions included in this
7Package using the inclusion mechanism, may be associated with other
8namespaces.

9The elements informationType, token, tokenLocator, roleType, relationshipType,
10participantType and channelType MAY be used as elements by all the
11Choreographies defined within this Package.

122.2.2 Choreography document Naming and Linking

13WS-CDL documents MUST be assigned a name attribute of type NCNAME that
14serves as a lightweight form of documentation.

15The targetNamespace attribute of type URI MUST be specified.

16The URI MUST NOT be a relative URI.

17A reference to a definition is made using a QName.

18Each definition type has its own name scope.

19Names within a name scope MUST be unique within a WS-CDL document.

20The resolution of QNames in WS-CDL is similar to the resolution of QNames
21described by the XML Schemas specification [11].

222.2.3 Language Extensibility and Binding

23To support extending the WS-CDL language, this specification allows the use of
24extensibility elements and/or attributes defined in other XML namespaces.

25Extensibility elements and/or attributes MUST use an XML namespace different
26from that of WS-CDL. All extension namespaces used in a WS-CDL document
27MUST be declared.

28Extensions MUST NOT change the semantics of any element or attribute from
29the WS-CDL namespace.

302.2.4 Semantics

31Within a WS-CDL document, descriptions allow the recording of semantics
32definitions and other documentation. The optional description sub-element is
33allowed inside any WS-CDL language element. WS-CDL parsers are not
34required to parse the contents of the description.

1The information provided by the description element will allow for the recording
2of semantics in any or all of the following ways:

- 3 • *Text*. This will be in plain text or possibly HTML and should be brief
- 4 • *Document Reference*. This will contain a URI to a document that more
5 fully describes the component. For example on the top level
6 Choreography Definition that might reference a complete paper
- 7 • *Machine Oriented Semantic Descriptions*. This will contain machine
8 processable definitions in languages such as RDF or OWL

9Descriptions that are text or document references can be defined in multiple
10different human readable languages.

112.3 Collaborating Parties

12The WSDL specification [7] describes the functionality of a service provided by a
13party based on a stateless, client-server model. The emerging Web Based
14applications require the ability to exchange information in a peer-to-peer
15environment. In these types of environments a party represents a requester of
16services provided by another party and is at the same time a provider of services
17requested from other parties, thus creating mutual multi-party service
18dependencies.

19A WS-CDL document describes how a party is capable of engaging in
20collaborations with the same party or with different parties.

21The *Role Types*, *Participant Types*, *Relationship Types* and *Channel Types*
22define the coupling of the collaborating parties.

232.3.1 Role Types

24*Role Type* enumerates the observable behavior a party exhibits in order to
25collaborate with other parties. For example the Buyer Role Type is associated
26with purchasing of goods or services and the Supplier Role Type is associated
27with providing those goods or services for a fee.

28

29The syntax of the *roleType* construct is:

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

34The attribute *name* is used for specifying a distinct name for each *roleType* element
35declared within a Choreography Package.

36Within the *roleType* element, the *behavior* element specifies a subset of the
37observable behavior a party exhibits. A Role Type MUST contain one or more
38behavior elements.

1The behavior element defines an optional interface attribute, which identifies a
2WSDL interface type. A behavior without an interface describes a Role Type that is
3not required to support a specific Web Service interface.

42.3.2 Relationship Types

5A *Relationship Type* identifies the Role Type and Behaviors where mutual
6commitments between two parties **MUST** be made for them to collaborate
7successfully. For example the Relationship Types between a Buyer and a Seller
8could include:

- 9 • A "Purchasing" Relationship Type, for the initial procurement of goods
10 or services, and
- 11 • A "Customer Management" Relationship Type to allow the Supplier to
12 provide service and support after the goods have been purchased or
13 the service provided

14Although Relationship Types are always between two Role Types,
15Choreographies involving more than two Role Types are possible. For example if
16the purchase of goods involved a third-party Shipper contracted by the Supplier
17to deliver the Supplier's goods, then, in addition to the Purchasing and Customer
18Management Relationship Types described above, the following Relationship
19Types might exist:

- 20 • A "Logistics Provider" Relationship Type between the Supplier and the
21 Shipper, and
- 22 • A "Goods Delivery" Relationship Type between the Buyer and the
23 Shipper

24

25The syntax of the *relationshipType* construct is:

```
27 <relationshipType name="ncname">  
28   <role type="qname" behavior="list of ncname"? />  
29   <role type="qname" behavior="list of ncname"? />  
30 </relationshipType>
```

31The attribute name is used for specifying a distinct name for each *relationshipType*
32element declared within a Choreography Package.

33A *relationshipType* element **MUST** have exactly two Role Types defined. Each Role
34Type is specified by the *role* element.

35Within the *role* element, the optional attribute *behavior* identifies the commitment of
36a party as a list of behavior types belonging to the Role Type specified by the
37*type* attribute of the *role* element. If the *behavior* attribute is missing then all the
38behaviors belonging to the Role Type specified by the *type* attribute of the *role*
39element are identified as the commitment of a party.

12.3.3 Participant Types

2A *Participant Type* identifies a set of Role Types that MUST be implemented by
3the same entity or organization. Its purpose is to group together the parts of the
4observable behavior that MUST be implemented by the same entity or
5organization.

6

7The syntax of the *participantType* construct is:

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

12The attribute name is used for specifying a distinct name for each participantType
13element declared within a Choreography Package.

14Within the participantType element, one or more role elements identify the Role
15Types that MUST be implemented by this Participant Type. Each Role Type is
16specified by the type attribute of the role element.

17

18An example is given below where the “SellerForBuyer” Role Type belonging to a
19“Buyer-Seller” Relationship Type is implemented by the Participant Type “Broker”
20which also implements the “SellerForShipper” Role Type belonging to a “Seller-
21Shipper” Relationship Type:

22

```
23 <roleType name="Buyer">  
24   ...  
25 </roleType>  
26 <roleType name="SellerForBuyer">  
27   <behavior name="sellerForBuyer" interface="rns:sellerForBuyerPT"/>  
28 </roleType>  
29 <roleType name="SellerForShipper">  
30   <behavior name="sellerForShipper" interface="rns:sellerForShipperPT"/>  
31 </roleType>  
32 <roleType name="Shipper">  
33   ...  
34 </roleType>  
35 <relationshipType name="Buyer-Seller">  
36   <role type="tns:Buyer" />  
37   <role type="tns:SellerForBuyer" />  
38 </relationshipType>  
39 <relationshipType name="Seller-Shipper">  
40   <role type="tns:SellerForShipper" />  
41   <role type="tns:Shipper" />  
42 </relationshipType>  
43  
44 <participantType name="Broker">  
45   <role type="tns:SellerForBuyer" />  
46   <role type="tns:SellerForShipper" />  
47 </participantType>
```

12.3.4 Channel Types

2A *Channel* realizes a point of collaboration between parties by specifying where
3and how information is exchanged between collaborating parties. Additionally,
4Channel information can be passed among parties in an information exchange.
5The Channels exchanged MAY be used in subsequent Interaction activities. This
6allows the modeling of both static and dynamic message destinations when
7collaborating within a Choreography. For example, a Buyer could specify
8Channel information to be used for sending delivery information. The Buyer
9could then send the Channel information to the Seller who then forwards it to the
10Shipper. The Shipper could then send delivery information directly to the Buyer
11using the Channel information originally supplied by the Buyer.

12A Channel Type MUST describe the Role Type and the reference type of a
13party, being the target of an information exchange, which is then used for
14determining where and how to send or receive information to or into the party.

15A Channel Type MAY specify the instance identity of an entity implementing the
16behavior(s) of a party, being the target of an information exchange.

17A Channel Type MAY describe one or more logical conversations between
18parties, where each conversation groups a set of related information exchanges.

19One or more Channel(s) MAY be passed around from one party to another in an
20information exchange. A Channel Type MAY be used to:

- 21 • Restrict the number of times a Channel of this Channel Type can be
22 used
- 23 • Restrict the type of information exchange that can be performed when
24 using a Channel of this Channel Type
- 25 • Restrict the Channel Type(s) that will be passed through a Channel of
26 this Channel Type
- 27 • Enforce that a passed Channel is always distinct

28

29The syntax of the *channelType* construct is:

```
31 <channelType name="ncname"  
32     usage="once"|"unlimited"?  
33     action="request-respond"|"request"|"respond"? >  
34  
35     <passing channel="qname"  
36         action="request-respond"|"request"|"respond"?  
37         new="xsd:boolean"? /*  
38  
39     <role type="qname" behavior="ncname"? /*  
40  
41     <reference>  
42         <token type="qname"/*  
43     </reference>  
44
```

```
1 <identity>
2 <token type="qname"/>+
3 </identity>?
4 </channelType>
```

5The attribute name is used for specifying a distinct name for each channelType
6element declared within a Choreography Package.

7The optional attribute usage is used to restrict the number of times a Channel of
8this Channel Type can be used.

9The optional attribute action is used to restrict the type of information exchange
10that can be performed when using a Channel of this Channel Type. The type of
11information exchange performed could either be a request-respond exchange, a
12request exchange, or a respond exchange. The default for this attribute is set to
13“request-respond”.

14The optional element passing describes the Channel Type(s) of the Channel(s)
15that are passed from one party to another, when using in an information
16exchange a Channel of this Channel Type. The optional attribute action within the
17passing element defines if a Channel will be passed during a request exchange,
18during a response exchange or both. The default for this attribute is set to
19“request-respond”. The optional attribute new within the passing element when set
20to “true” enforces a passed Channel to be always distinct.

21If the element passing is missing then this Channel Type MAY be used for
22exchanging business documents and for passing Channels of any Channel Type
23without any restrictions.

24The element role is used to identify the Role Type of a party, being the target of
25an information exchange, which is then used for statically determining where and
26how to send or receive information to or into the party.

27The element reference is used for describing the reference type of a party, being
28the target of an information exchange, which is then used for dynamically
29determining where and how to send or receive information to or into the party.
30The reference of a party is distinguished by a Token as specified by the token
31element within the reference element.

32The optional element identity MAY be used for identifying an instance of an entity
33implementing the behavior of a party and for identifying a logical conversation
34between parties. The identity and the different conversations are distinguished
35by a set of Tokens as specified by the token element within the identity element.

36

37The following rule applies for Channel Type:

- 38 • If two or more Channel Types SHOULD point to Role Types that MUST
39 be implemented by the same entity or organization, then the specified
40 Role Types MUST belong to the same Participant Type. In addition the
41 identity elements within the Channel Types MUST have the same
42 number of Tokens with the same informationTypes specified in the
43 same order

1

2The example below shows the definition of the Channel Type “RetailerChannel”
3that realizes a point of collaboration with a Retailer. The Channel Type identifies
4the Role Type of the Retailer as the “Retailer”. The information for locating the
5Retailer is specified in the reference element, whereas the instance of a process
6implementing the Retailer is identified for correlation purposes using the identity
7element. The element passing allows only a Channel of “ConsumerChannel” Type
8to be passed in a request information exchange thought a Channel of
9“RetailerChannel” Type.

```
11 <channelType name="RetailerChannel">
12   <passing channel="ConsumerChannel" action="request" />
13
14   <role type="tns:Retailer" behavior="retailerForConsumer"/>
15
16   <reference>
17     <token type="tns:retailerRef"/>
18   </reference>
19
20   <identity>
21     <token type="tns:purchaseOrderID"/>
22   </identity>
23 </channelType>
```

242.4 Information Driven Collaborations

25Parties make progress within a collaboration, when recordings of exchanged
26information and observable information changes cause ordering constraints to
27be fulfilled. A WS-CDL document allows defining information within a
28Choreography that can influence the observable behavior of the collaborating
29parties.

30Variables capture information about objects in the Choreography, such as the
31information exchanged or the observable information of the Roles involved.
32Token are aliases that can be used to reference parts of a Variable. Both
33Variables and Tokens have *Information Types* that define the type of information
34the Variable or Token contain.

352.4.1 Information Types

36Information types describe the type of information used within a Choreography.
37By introducing this abstraction, a Choreography definition avoids referencing
38directly the data types, as defined within a WSDL document or an XML Schema
39document.

40

41The syntax of the *informationType* construct is:

```
43 <informationType name="ncname"
44   type="qname"? | element="qname"?
```

1 `exceptionType="true|false"/>`

2The attribute name is used for specifying a distinct name for each informationType
3element declared within a Choreography Package.

4The attributes type, and element describe the type of information used within a
5Choreography as a WSDL 1.1 Message Type, an XML Schema type, a WSDL
62.0 Schema element or an XML Schema element. The type of information is of
7one of these types exclusively.

8When the attribute exceptionType is set to "true", this information type is an
9Exception Type and could map to WSDL fault type. By default, this attribute is
10set to "false".

11In case of WSDL 2.0, the attribute element within the informationType refers to a
12unique WSDL 2.0 faultname when the attribute exceptionType is set to "true".

13

14The examples below show some possible usages of informationType.

15

16 **Example1:** The informationType "purchaseOrder" refers to the WSDL 1.1 Message type
17 "pns:purchaseOrderMessage"
18 `<informationType name="purchaseOrder" type="pns:purchaseOrderMessage"/>`
19
20
21 **Example2:** The informationType "customerAddress" refers to the WSDL 2.0 Schema element
22 "cns:CustomerAddress"
23 `<informationType name="customerAddress" element="cns:CustomerAddress"/>`
24
25
26 **Example 3:** The informationType "intType" refers to the XML Schema type "xsd:int"
27 `<informationType name="intType" type="xsd:int"/>`
28
29
30
31 **Example 4:** The informationType "OutOfStockExceptionType" is of type Exception Type and
32 refers to the WSDL 2.0 fault name "cwns:OutOfStockExceptionType"
33
34 `<informationType name="OutOfStockExceptionType"`
35 `type="cwns:OutOfStockExceptionType" exceptionType="true"/>`

362.4.2 Variables

37Variables capture information about objects in a Choreography as defined by
38their usage:

- 39 • *Information Exchange Capturing Variables*, which contain information
40 such as an Order that is used to
- 41 • Populate the content of a message to be sent, or

20

- 1 • Populated as a result of a message received
- 2 • *State Capturing Variables*, which contain information about the
3 observable changes of a Role as a result of information being
4 exchanged. For example when a Buyer sends an Order to a Seller, the
5 Buyer could have a Variable called "OrderState" set to a value of
6 "OrderSent" and once the message was received by the Seller, the
7 Seller could have a Variable called "OrderState" set to a value of
8 "OrderReceived". Note that the Variable "OrderState" at the Buyer is a
9 different Variable to the "OrderState" at the Seller
- 10 • *Channel Capturing Variables*. For example, a Channel Variable could
11 contain information such as the URL to which the message could be
12 sent, the policies that are to be applied, such as security, whether or
13 not reliable messaging is to be used, etc.

14 The value of Variables:

- 15 • Is available to Roles within a Choreography, when the Variables
16 contain information that is common knowledge. For example the
17 Variable "OrderResponseTime" which is the time in hours in which a
18 response to an Order must be sent is initialized prior to the initiation of
19 a Choreography and can be used by all Roles within the Choreography
- 20 • Can be made available as a result of an Interaction
- 21 • Information Exchange Capturing Variables are populated and
22 become available at the Roles in the ends of an Interaction
- 23 • State Capturing Variables, that contain information about the
24 observable information changes of a Role as a result of
25 information being exchanged, are recorded and become
26 available
- 27 • Can be created or changed and made available locally at a Role by
28 assigning data from other information. They can be Information
29 Exchange, State or Channel Capturing Variables. For example
30 "Maximum Order Amount" could be data created by a Seller that is
31 used together with an actual order amount from an Order received to
32 control the ordering of the Choreography. In this case how "Maximum
33 Order Amount" is calculated and its value would not be known by the
34 other Roles
- 35 • Can be used to determine the decisions and actions to be taken within
36 a Choreography
- 37 • Can be used to cause Exceptions at one or more parties in a
38 Choreography

39 The *variableDefinitions* construct is used for defining one or more Variables
40 within a Choreography.

41

1 The syntax of the *variableDefinitions* construct is:

```
3 <variableDefinitions>
4   <variable   name="ncname"
5     informationType="qname" | channelType="qname "
6     mutable="true|false"?
7     free="true|false"?
8     silent="true|false"?
9     roleType="qname"? />+
10 </variableDefinitions>
```

11 The defined Variables can be of the following types:

- 12 • Information Exchange Capturing Variables, State Capturing Variables,
13 Exception Capturing Variables. The attribute *informationType* describes
14 the type of the object captured by the Variable
- 15 • Channel Capturing Variables. The attribute *channelType* describes the
16 type of the channel object captured by the Variable

17 The optional attribute *mutable*, when set to "false" describes that the Variable
18 information when initialized, cannot change anymore. The default value for this
19 attribute is "true".

20 The optional attribute *free*, when set to "true" describes that a Variable defined in
21 an enclosing Choreography is also used in this Choreography, thus sharing the
22 Variables information. The following rules apply in this case:

- 23 • The type (as specified by the *informationType* or the *channelType* attributes)
24 of a free Variable MUST match the type of the Variable defined in an
25 enclosing Choreography
- 26 • The attributes *silent* and *mutable* of a free Variable MUST match the
27 attributes *silent* and *mutable* of the Variable defined in an enclosing
28 Choreography
- 29 • A perform activity MUST bind a free Variable defined in an enclosed
30 Choreography with a Variable defined in an enclosing Choreography

31 The optional attribute *free*, when set to "false" describes that a Variable is defined
32 in this Choreography.

33 The default value for the *free* attribute is "false".

34 The optional attribute *silent*, when set to "true" describes that there SHOULD NOT
35 be any activity used for creating or changing this Variable in the Choreography, if
36 these operations should not be observable to other parties. The default value for
37 this attribute is "false".

38 The optional attribute *roleType* is used to specify the Role Type of a party at which
39 the Variable information will reside.

40 The following rules apply to Variable Definitions:

- 41 • The attribute *name* is used for specifying a distinct name for each
42 variable element declared within a *variableDefinitions* element when

- 1 needed. The Variables with Role Type not specified MUST have
2 distinct names. The Variables with Role Type specified MUST have
3 distinct names, when their Role Type is the same
- 4 • A Variable defined without a Role Type is equivalent to a Variable that
5 is defined at all the Role Types that are part of the Relationship Types
6 of the Choreography where the Variable is defined. For example if
7 Choreography C1 has Relationship Type R that has a tuple (Role1,
8 Role2), then a Variable "var" defined in Choreography C1 without a
9 roleType attribute means it is defined at Role1 and Role2
 - 10 • A Variable defined with informationType having the attribute exceptionType
11 set to "true" is an *Exception Capturing Variable*

122.4.3 Expressions

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

15 Generic expressions and literals can be used for populating a Variable.

16 Predicate expressions can be used within WS-CDL to specify conditions. Query
17 expressions are used within WS-CDL to specify query strings.

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

20 WS-CDL defines XPath function extensions as described in the following
21 section. The function extensions are defined in the standard WS-CDL
22 namespace "http://www.w3.org/ws/choreography/2004/09/WSCDL". The prefix "cdl:" is
23 associated with this namespace.

242.4.3.1 WS-CDL Supplied Functions

25 There are several functions that the WS-CDL specification supplies as XPATH
26 1.0 extension functions. These functions can be used in any XPath expression
27 as long as the types are compatible: *xsd:time getCurrentTime(xsd:QName
28 roleName)*.

29 Returns the current time at the caller for the Role specified by *roleName* (i.e. a
30 role can ask only about it's own time).

31

32 *xsd:date getCurrentDate(xsd:QName roleName)*.

33 Returns the current date at the caller for the Role specified by *roleName* (i.e. a
34 role can ask only about it's own date).

35

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

37 Returns the current date and time at the caller for the Role specified by
38 *roleName* (i.e. a role can ask only about it's own date/time).

1

2 *xsd:boolean hasTimeElapsed(xsd:duration elapsedTime, xsd:QName
3 roleName).*

4 Returns "true" if used in a guard or repetition condition of a Work Unit with the
5 block attribute set to "true" and the time specified by elapsedTime at the caller for
6 the Role specified by roleName has elapsed from the time the either the guard or
7 the repetition condition were enabled for matching. Otherwise it returns "false".

8

9 *xsd:string createNewID()*

10 Returns a new globally unique string value for use as an identifier.

11

12 *xsd:any getVariable(xsd:string varName, xsd:string part, xsd:string
13 documentPath, xsd:QName roleName?)*

14 Returns the information of the Variable with name varName as a node set
15 containing a single node. The second parameter, part, specifies the message
16 part of a WSDL 1.1 document. For a WSDL 2.0 document it MUST be empty.
17 When the third parameter documentPath is empty, then this function retrieves
18 the entire document from the Variable information. When it is non-empty, then
19 this function retrieves from the Variable information, the fragment of the
20 document at the provided absolute location path. The fourth parameter is
21 optional. When the fourth parameter is used that the Variable information MUST
22 be available at the Role specified by roleName. If this parameter is not used then
23 the Role is inferred from the context that this function is used.

24

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

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

28

29 *xsd:boolean variablesAligned(xsd:string varName, xsd:string withVarName,
30 xsd:QName relationshipName)*

31 Returns "true" if within a Relationship specified by relationshipName the Variable
32 with name varName residing at the first Role of the Relationship has aligned its
33 information with the Variable named withVarName residing at the second Role of
34 the Relationship.

35

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

37 Returns the reference information of the Variable with name varName. The
38 Variable MUST be of Channel Type.

39

24

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

2 Returns the identity information of the Variable with name *varName*. The

3 Variable MUST be of Channel Type.

4

5 *xsd:boolean globalizedTrigger(xsd:string expression, xsd:string roleName,*
6 *xsd:string expression2, xsd:string roleName2, ...)*

7 Combines expressions that include Variables that are defined at different Roles.

8

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

10 Returns "true" if an exception of Exception Type identified by the parameter

11 *exceptionType* has occurred. Otherwise it returns "false". The informationType with

12 name *exceptionType* MUST have *exceptionType* attribute set to "true".

13 2.4.4 Tokens

14 A *Token* is an alias for a piece of data in a Variable or message that needs to be

15 used by a Choreography. Tokens differ from Variables in that Variables contain

16 values whereas Tokens contain information that define the piece of the data that

17 is relevant. For example a Token for "Order Amount" within an Order XML

18 document could be an alias for an expression that pointed to the Order Amount

19 element within the Order XML document. This could then be used as part of a

20 condition that controls the ordering of a Choreography, for example "Order

21 Amount > \$1000".

22 All Tokens MUST have an informationType, for example, an "Order Amount"

23 would be of type "amount", "Order Id" could be alphanumeric and a counter an

24 integer.

25 Tokens types reference a document fragment within a Choreography definition

26 and Token Locators provide a query mechanism to select them. By introducing

27 these abstractions, a Choreography definition avoids depending on specific

28 message types, as described by WSDL, or a specific query string, as specified

29 by XPATH, but instead the document part and the query string can change

30 without affecting the Choreography definition.

31

32 The syntax of the *token* construct is:

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

35 The attribute *name* is used for specifying a distinct name for each token element

36 declared within a Choreography Package.

37 The attribute *informationType* identifies the type of the document fragment.

38

39 The syntax of the *tokenLocator* construct is:

```

1 <tokenLocator tokenName="qname"
2     informationType="qname"
3     part="ncname"?
4     query="XPath-expression" />

```

5The attribute `tokenName` identifies the name of the Token that the document
6fragment locator is associated with.

7The attribute `informationType` identifies the type of the document on which the
8query is performed to locate the Token.

9The optional attribute `part` defines the document part on which the query is
10performed to locate the Token. This attribute SHOULD NOT be defined for a
11WSDL 2.0 document.

12The attribute `query` defines the query string that is used to select a document
13fragment within a document or a document part.

14

15The example below shows that the Token "purchaseOrderID" is of type `xsd:int`.
16The two `tokenLocators` show how to access this token in "purchaseOrder" and
17"purchaseOrderAck" messages.

```

19 <token name="purchaseOrderID" informationType="xsd:int"/>
20 <tokenLocator tokenName="tns:purchaseOrderID" informationType="purchaseOrder"
21     query="/PO/OrderId"/>
22 <tokenLocator tokenName="tns:purchaseOrderID" informationType="purchaseOrderAck"
23     query="/POAck/OrderId"/>

```

242.4.5 Choreographies

25A *Choreography*. defines re-usable common rules, that govern the ordering of
26exchanged messages and the provisioning patterns of collaborative behavior,
27agreed between two or more interacting parties

28A Choreography defined at the Package level is called a *top-level* Choreography,
29and does not share its context with other top-level Choreographies. A Package
30MAY contain exactly one top-level Choreography, marked explicitly as the *root*
31Choreography. The root Choreography is the only top-level Choreography that is
32enabled by default.

33A Choreography defines the re-usable the common rules, that govern the
34ordering of exchanged messages and the provisioning patterns of behavior,
35action(s) performing the actual work, such as exchange of information, when the
36specified ordering constraints are satisfied.

37The re-usable behavior encapsulated within a Choreography MAY be performed
38within an *enclosing* Choreography, thus facilitating composition. The performed
39Choreography is then called an *enclosed* Choreography.

40The Choreography that is performed MAY be defined:

- 1 • *Locally* - its definition is contained within the Choreography definition of
2 the Choreography that performed it
- 3 • *Globally* - a separate top-level Choreography definition is specified in
4 the same or in a different Choreography Package that can be used by
5 other Choreographies and hence the contract is reusable

6A non-root Choreography is enabled when performed.

7A Choreography MUST contain at least one Relationship Type, enumerating the
8observable behavior this Choreography requires its parties to exhibit. One or
9more Relationship Types MAY be defined within a Choreography, modeling
10multi-party collaborations.

11A Choreography acts as a lexical name scoping context for Variables. A Variable
12defined in a Choreography is visible for use in this Choreography and all its
13enclosed Choreographies up-to the point that the Variable is re-defined as an
14non-free Variable, thus forming a *Choreography Visibility Horizon* for this
15Variable.

16A Choreography MAY contain one or more Choreography definitions that MAY
17be performed only locally within this Choreography.

18A Choreography MUST contain an *activity*. The activity specifies the enclosed
19actions of the Choreography that perform the actual work. The enclosed actions
20specified by the activity are enabled when the Choreography they belong to is
21enabled.

22A Choreography can recover from exceptional conditions and provide finalization
23actions by defining:

- 24 • One *Exception Block*, which MAY be defined as part of the
25 Choreography to recover from exceptional conditions that can occur
- 26 • One *Finalizer Block*, which MAY be defined as part of the
27 Choreography to provide the finalization actions for that Choreography

28

29The *choreography* is used to define a Choreography. The syntax is:

```

31 <choreography name="ncname"
32     complete="xsd:boolean XPath-expression"?
33     isolation="dirty-write|"dirty-read|"serializable"?
34     root="true|"false"? >
35
36     <relationship type="qname" />+
37
38     variableDefinitions?
39
40     choreography*
41
42     activity
43
44     <exception name="ncname">
45         workunit+

```

```
1     </exception>?  
2     <finalizer name="ncname">  
3         workunit  
4     </finalizer>?  
5 </choreography>
```

6The attribute name is used for specifying a distinct name for each choreography
7element declared within a Choreography Package.

8The optional complete attribute allows to explicitly complete a Choreography as
9described below in the Choreography Life-line section.

10The optional isolation attribute specifies when Variable information that is defined
11in an enclosing, and changed within an enclosed Choreography is available to its
12sibling Choreographies:

- 13 • When isolation is set to "dirty-write", the Variable information MAY be
14 immediately overwritten by actions in other Choreographies
- 15 • When isolation is set to "dirty-read", the Variable information MAY be
16 immediately visible for read but not for write to other Choreographies
- 17 • When isolation is set to "serializable", the Variable information MUST be
18 visible for read or for write to other Choreographies only after this
19 Choreography has ended successfully

20The relationship element within the choreography element enumerates the
21Relationships this Choreography MAY participate in.

22The optional variableDefinitions element enumerates the Variables defined in this
23Choreography.

24The optional root element marks a top-level Choreography as the root
25Choreography of a Choreography Package.

26The optional choreography within the choreography element defines the Locally
27defined Choreographies that MAY be performed only within this Choreography.

28The optional exception element defines the Exception Block of a Choreography by
29specifying one or more Exception Work Unit(s). Within this element, the attribute
30name is used for specifying a name for this Exception Block element.

31The optional finalizer element defines the Finalizer Block of a Choreography by
32specifying one Finalizer Work Unit. Within this element, the attribute name is used
33for specifying a name for this Finalizer Block element.

342.4.6 WorkUnits

35A *Work Unit* prescribes the constraints that must be fulfilled for making progress
36and thus performing actual work within a Choreography. Examples of a Work
37Unit include:

- 38 • A *Send PO* Work Unit that includes Interactions for the Buyer to send
39 an Order, the Supplier to acknowledge the order, and then later accept

- 1 (or reject) the Order. This Work Unit would probably not have a guard
2 condition
- 3 • An *Order Delivery Error* Work Unit that is performed whenever the
4 *Place Order* Work Unit did not reach a "normal" conclusion. This would
5 have a guard condition that identifies the error
 - 6 • A *Change Order* Work Unit that can be performed whenever an order
7 acknowledgement message has been received and an order rejection
8 has not been received

9 A Work Unit MAY prescribe the constraints that preserve the consistency of the
10 collaborations commonly performed between the parties. Using a Work Unit an
11 application MAY recover from errors that are the result of abnormal actions and
12 also MAY finalize completed actions that need to be logically rolled back.

13 When enabled, a Work Unit expresses interest(s) on the availability of one or
14 more Variable information that already exist or will be created in the future.

15 The Work Unit's interest(s) are matched when all the required Variable
16 information is available or has become available and the specified matching
17 guard condition on the Variable information is met. Variable information available
18 within a Choreography MAY be matched with a Work Unit that will be enabled in
19 the future. One or more Work Units MAY be matched concurrently if their
20 respective interests are matched. When a Work Unit matching succeeds then its
21 enclosed actions are enabled.

22 A Work Unit MUST contain an activity that performs the actual work.

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

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

28

29 The *Work Unit* is defined as follows:

```
31 <workunit name="ncname"  
32     guard="xsd:boolean XPath-expression"?  
33     repeat="xsd:boolean XPath-expression"?  
34     block="true|false"? >  
35     activity  
36 </workunit>
```

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

40 The activity specifies the enclosed actions of a Work Unit.

41 The guard condition of a Work Unit, specified by the optional guard attribute,
42 describes the interest on the availability of one or more, existing or future
43 Variable information.

1The optional repeat attribute allows, when the condition it specifies evaluates to
2"true", to make the current Work Unit that completed successfully to be
3considered again for matching (based on the guard attribute).

4The optional attribute block specifies whether the matching condition relies on the
5Variable that is currently available, or whether the Work Unit has to block waiting
6for the Variable to become available in the future if it is not currently available.
7This attribute MUST always be set to "false" in Exception Work Units. The
8default for this attribute is set to "false".

9

10The following rules apply:

- 11 • When a guard condition is not specified then the Work Unit always
12 matches
- 13 • When a repetition condition is not specified then the Work Unit is not
14 considered again for matching after the Work Unit got matched once
- 15 • One or more Variables can be specified in a guard condition or
16 repetition condition, using XPATH and the WS-CDL functions, as
17 described in Section 2.4.3
- 18 • The WS-CDL function getVariable is used in the guard or repetition
19 condition to obtain the information of a Variable
- 20 • When the WS-CDL function isVariableAvailable is used in the guard or
21 repetition condition, it means that the Work Unit that specifies the
22 guard condition is checking if a Variable is already available at a
23 specific Role or is waiting for a Variable to become available at a
24 specific Role, based on the block attribute being "false" or "true"
25 respectively
- 26 • When the WS-CDL function variablesAligned is used in the guard or
27 repetition condition, it means that the Work Unit that specifies the
28 guard or repetition condition is checking or waiting for an appropriate
29 alignment Interaction to happen between the two Roles, based on the
30 block attribute being "false" or "true" respectively. The Variables
31 checked or waited for alignment are the sending and receiving ones in
32 an alignment Interaction or the ones used in the recordings at the two
33 Roles in the ends of an alignment Interaction. When the variablesAligned
34 WS-CDL function is used in a guard or repetition condition, then the
35 Relationship Type within the variablesAligned MUST be the subset of the
36 Relationship Type that the immediate enclosing Choreography defines
- 37 • Variables defined at different Roles MAY be used in a guard condition
38 or repetition condition to form a *globalized* view that combines
39 constraints prescribed for each Role. The globalizedTrigger WS-CDL
40 function MUST be used in a guard condition or repetition condition in
41 this case. Variables defined at the same Role MAY be combined
42 together in a guard condition or repetition condition using all available

- 1 XPATH operators and all the WS-CDL functions except the
2 globalizedTrigger WS-CDL function
- 3 • If the attribute `block` is set to "true" and one or more required Variable(s)
4 are not available, then the Work Unit MUST block. When the required
5 Variable information specified by the guard condition become available
6 and the guard condition evaluates to "true", then the Work Unit is
7 matched. If the repetition condition is specified, then it is evaluated
8 when the Work Unit completes successfully. Then, if the required
9 Variable information specified by the repetition condition is available
10 and the repetition condition evaluates to "true", the Work Unit is
11 considered again for matching. Otherwise, the Work Unit is not
12 considered again for matching
 - 13 • If the attribute `block` is set to "false", then the guard condition or
14 repetition condition assumes that the Variable information is currently
15 available. If either the Variable information is not available or the guard
16 condition evaluates to "false", then the Work Unit matching fails and
17 the activity enclosed within the Work Unit is skipped and the repetition
18 condition even if specified is not evaluated. Otherwise, if the repetition
19 condition is specified, then it is evaluated when the Work Unit
20 completes successfully. Then, if the required Variable information
21 specified by the repetition condition is available and the repetition
22 condition evaluates to "true", the Work Unit is considered again for
23 matching. Otherwise, the Work Unit is not considered again for
24 matching
- 25

26 The examples below demonstrate the possible use of a Work Unit:

27a. *Example of a Work Unit with block equals to "true"*:

28 In the following Work Unit, the guard condition waits on the availability of
29 "POAcknowledgement" at "Customer" Role and if it is already available, the
30 activity happens, otherwise, the activity waits until the Variable
31 "POAcknowledgement" become available at the "Customer" Role.

```
33 <workunit name="POProcess"  
34     guard="cdl:isVariableAvailable(  
35         cdl:getVariable("POAcknowledgement"), "", "", "tns:customer")"  
36     block="true">  
37     ... <!--some activity -->  
38 </workunit>
```

39b. *Example of a Work Unit with block equals to "false"*:

40 In the following Work Unit, the guard condition checks if the Variable
41 "StockQuantity" at the "Retailer" Role is available and is greater than 10 and if
42 so, the activity happens. If either the Variable is not available or the value is less
43 than 10, the matching condition is "false" and the activity is skipped.

```

1 <workunit name="Stockcheck"
2     guard="cdl:getVariable("StockQuantity", "", "/Product/Qty",
3         "tns:retailer") > 10) "
4     block="false" >
5     ... <!--some activity -->
6 </workunit>

```

7c. Example of a Work Unit waiting for alignment to happen..

8In the following Work Unit, the guard condition waits for an alignment Interaction
9to happen between the “Customer” Role and the “Retailer” Role:

```

11 <workunit name="WaitForAlignment"
12     guard="cdl:variablesAligned(
13         "PurchaseOrderAtBuyer", "PurchaseOrderAtSeller",
14         "customer-retailer-relationship)"
15     block="true" >
16     ... <!--some activity -->
17 </workunit>

```

182.4.7 Including Choreographies

19Choreographies or fragments of Choreographies can be syntactically reused in
20any Choreography definition by using XInclude [27]. The assembly of large
21Choreography definitions from multiple smaller, well formed Choreographies or
22Choreography fragments is enabled using this mechanism.

23

24The example below shows a possible syntactic reuse of a Choreography
25definition:

```

27 <choreography name="newChoreography" root="true">
28     ...
29     <variable name="newVariable" informationType="someType"
30         role="randomRome"/>
31     <xi:include href="genericVariableDefinitions.xml" />
32     <xi:include href="otherChoreography.xml"
33         xpointer="xpointer(//choreography/variable[1]) />
34     ...
35 </choreography>

```

362.4.8 Choreography Life-line

37A Choreography life-line expresses the progression of a collaboration. Initially,
38the collaboration MUST be started, then work MAY be performed within it and
39finally it MAY complete. These different phases are designated by explicitly
40marked actions within the Choreography.

41

42A root Choreography is initiated when the first Interaction, marked as the
43Choreography initiator, is performed. Two or more Interactions MAY be marked

32

1as initiators, indicating alternative initiation actions. In this case, the first action
2will initiate the Choreography and the other actions will enlist with the already
3initiated Choreography. An Interaction designated as a Choreography initiator
4MUST be the first action performed in a Choreography. If a Choreography has
5two or more Work Units with Interactions marked as initiators, then these are
6mutually exclusive and the Choreography will be initiated when the first
7Interaction occurs and the remaining Work Units will be disabled. All the
8Interactions not marked as initiators indicate that they will enlist with an already
9initiated Choreography.

10A Choreography completes successfully when there are no more matched Work
11Unit(s) performing work within it and there are no enabled Work Unit(s) within it.
12Alternatively, a Choreography completes successfully if its complete condition,
13as defined by the optional complete attribute within the choreography element,
14evaluates to "true". In this case, the actions, including enclosed Choreographies,
15within the explicitly completed Choreography are completed abnormally before
16the Choreography completes.

172.4.9 Choreography Recovery

18One or more Exception WorkUnit(s) MAY be defined as part of a Choreography
19to recover from exceptional conditions that can occur in that Choreography.

20A Finalizer WorkUnit MAY be defined as part of a Choreography to provide the
21finalization actions that semantically "undo" that completed Choreography.

222.4.9.1 Exception Block

23A Choreography can sometimes fail as a result of an exceptional circumstance
24or error.

25An *Exception* is caused in the Choreography when an Exception Variable is
26populated in an Interaction activity with the attribute `causeException` set to "true".

27An Exception is propagated to all parties in the Choreography using explicitly
28modeled, *Exception Causing Interactions*. This causes the Choreography to
29enter the Exception state and its Exception Block to be enabled, if specified.

30

31Different types of errors are possible including this non-exhaustive list:

- 32 • *Interaction Failures*, for example the sending of a message did not
33 succeed
- 34 • *Protocol Based Exchange failures*, for example no acknowledgement
35 was received as part of a reliable messaging protocol [22]
- 36 • *Security failures*, for example a Message was rejected by a recipient
37 because the digital signature was not valid
- 38 • *Timeout errors*, for example an Interaction did not complete within the
39 required time

- 1 • *Validation Errors*, for example an XML order document was not well
2 formed or did not conform to its schema definition
- 3 • *Application "failures"*, for example the goods ordered were out of stock

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

7 One or more Exception Work Unit(s) MAY be defined within the Exception Block
8 of a Choreography. At least one Exception Work Unit MUST be defined as part
9 of the Exception Block of a Choreography. An Exception Work Unit MAY express
10 interest on Exception information using its guard condition on Exception Types
11 or Exception Variables. If no guard condition is specified, then the Exception
12 Work Unit is called the *Default Exception Work Unit* and expresses interest on
13 any type of Exception. Within the Exception Block of a Choreography there
14 MUST NOT be more than one Default Exception Work Unit. An Exception Work
15 Unit MUST always set its block attribute to "false" and MUST NOT define a
16 repetition condition.

17 An Exception Work Unit MAY have a guard condition using the WS-CDL function
18 `hasExceptionOccurred` or the WS-CDL function `globalizedTrigger` on Exception
19 Variables of the same Exception Type involving all Roles in the Choreography.

20 Exception Work Units are enabled when the Exception Block of the
21 Choreography they belong to is enabled. Enabled Exception Work Units in a
22 Choreography MAY behave as a mechanism to recover from Exceptions
23 occurring in this and its enclosed Choreographies.

24 Within the Exception Block of a Choreography only one Exception Work Unit
25 MAY be matched.

26

27 The rules for matching an Exception are:

- 28 • When an Exception Work Unit has a guard condition using the WS-CDL
29 function `hasExceptionOccurred(exceptionType)`, then it is matched when an
30 Exception Variable with Exception Type that matches the parameter
31 `exceptionType` is populated using an Exception Causing Interaction activity
- 32 • All the Exception Variables specified in a guard condition of an Exception
33 Work Unit using the WS-CDL function `getGlobalizedTrigger` MUST be of the
34 same Exception Type
- 35 • If an Exception is matched by the guard condition of an Exception Work
36 Unit, then the actions of the matched Work Unit are enabled. When two or
37 more Exception Work Units are defined then the order of evaluating their
38 guard conditions is based on the order that the Work Units have been
39 defined within the Exception Block
- 40 • If none of the guard condition(s) match, then if there is a Default
41 Exception Work Unit without a guard condition defined then its actions are
42 enabled

34

- 1 • If an Exception is not matched by an Exception Work Unit defined within
2 the Choreography in which the Exception occurs, the Exception will be
3 recursively propagated to the Exception Work Unit of the immediate
4 enclosing Choreography until a match is successful
- 5 • If an Exception occurs within a Choreography, then the Choreography
6 completes unsuccessfully and this causes its Finalizer WorkUnit to be
7 disabled. The actions, including enclosed Choreographies, within the
8 Choreography are completed abnormally before an Exception Work Unit
9 can be matched

10The actions within the Exception Work Unit MAY use Variable information visible
11in the Visibility Horizon of the Choreography it belongs to as they stand at the
12current time.

13The actions of an Exception Work Unit MAY also cause an Exception. The
14semantics for matching the Exception and acting on it are the same as described
15in this section.

16**2.4.9.2 Finalizer Block**

17When a Choreography encounters an exceptional condition it MAY need to
18revert the actions it had already completed, by providing finalization actions that
19semantically rollback the effects of the completed actions. To handle these a
20separate Finalizer Work Unit is defined in the Finalizer Block of a Choreography.

21A Choreography MAY define one Finalizer Work Unit.

22A Finalizer WorkUnit is enabled only after the Choreography it belongs to
23completes successfully. The Finalizer Work Unit MAY be enabled only once.

24The actions within the Finalizer Work Unit MAY use Variable information visible
25in the Visibility Horizon of the Choreography it belongs to as they were at the
26time the Choreography completed for the Variables belonging to this
27Choreography or as they stand at the current time for the Variables belonging to
28the enclosing Choreography.

29The actions of the Finalizer Work Unit MAY cause an Exception. The semantics
30for matching this Exception and acting on it are the same as described in the
31previous section.

32**2.5 Activities**

33*Activities* are the lowest level components of the Choreography, used to describe
34the actual work performed when the specified ordering constraints are satisfied.
35

36An *activity* is then either:

37

- 38 • An *Ordering Structure* – which combines Activities with other Ordering
39 Structures in a nested way to specify the ordering rules of activities
40 within the Choreography

- 1 • A *WorkUnit*
- 2 • A *Basic Activity* that performs the actual work. A Basic Activity is then
3 either:
 - 4 • *Interaction*, which results in an exchange of information
5 between parties and possible synchronization of their
6 observable information changes and the actual values of the
7 exchanged information
 - 8 • A *Perform*, which means that a complete, separately defined
9 Choreography is performed
 - 10 • An *Assign*, which assigns, within one Role, the value of one
11 Variable to the value of another Variable
 - 12 • A *Silent Action*, which provides an explicit designator used for
13 specifying the point where party specific operation(s) with non-
14 observable operational details MUST be performed
 - 15 • A *No Action*, which provides an explicit designator used for
16 specifying the point where a party does not perform any action

172.5.1 Ordering Structures

18An *Ordering Structure* is one of the following:

- 19 • *Sequence*
- 20 • *Parallel*
- 21 • *Choice*

222.5.1.1 Sequence

23The *sequence* ordering structure contains one or more activities. When the
24sequence activity is enabled, the sequence element restricts the series of
25enclosed activities to be enabled sequentially, in the same order that they are
26defined.

27

28The syntax of this construct is:

```
30 <sequence>  
31     activity+  
32 </sequence>
```

332.5.1.2 Parallel

34The *parallel* ordering structure contains one or more activities that are enabled
35concurrently when the parallel activity is enabled. The parallel activity completes
36successfully when all activities performing work within it complete successfully.

37

1The syntax of this construct is:

```
3 <parallel>
4   activity+
5 </parallel>
```

62.5.1.3 Choice

7The *choice* ordering structure enables a Work Unit to define that only one of two
8or more activities SHOULD be performed.

9When two or more activities are specified in a choice element, only one activity is
10selected and the other activities are disabled. If the choice has Work Units with
11guard conditions, the first Work Unit that matches the guard condition is selected
12and the other Work Units are disabled. If the choice has other activities, it is
13assumed that the selection criteria for the activities are non-observable.

14

15The syntax of this construct is:

```
17 <choice>
18   activity+
19 </choice>
```

20

21In the example below, choice element has two Interactions, “processGoodCredit”
22and “processBadCredit”. The Interactions have the same directionality,
23participate within the same Relationship and have the same fromRoles and
24toRoles names. If one Interaction happens, then the other one is disabled.

```
26 <choice>
27   <interaction channelVariable="doGoodCredit-channel" operation="doCredit">
28     ...
29   </interaction>
30
31   <interaction channelVariable="badCredit-channel" operation="doBadCredit">
32     ...
33   </interaction>
34 </choice>
```

352.5.2 Interacting

36An *Interaction* is the basic building block of a Choreography, which results in
37information exchanged between collaborating parties and possibly the
38synchronization of their observable information changes and the values of the
39exchanged information.

40An Interaction forms the base atom of the Choreography composition, where
41multiple Interactions are combined to form a Choreography, which can then be
42used in different business contexts.

1An Interaction is initiated when one of the Roles participating in the Interaction
2sends a message, through a common Channel, to another Role that is
3participating in the Interaction, that receives the message. If the initial message
4is a request, then the accepting Role can optionally respond with a normal
5response message or a fault message, which will be received by the initiating
6Role.

7

8An Interaction also contains "references" to:

- 9 • The *Channel Capturing Variable* that specifies the interface and other
10 data that describe where and how the message is to be sent
- 11 • The *Operation* that specifies what the recipient of the message should
12 do with the message when it is received
- 13 • The *From Role* and *To Role* that are involved
- 14 • The *Information Type* or *Channel Type* that is being exchanged
- 15 • The *Information Exchange Capturing Variables* at the From Role and
16 To Role that are the source and destination for the message content
- 17 • A list of potential observable information changes that MAY occur and
18 MAY need to be aligned at the *From Role* and the *To Role*, as a result
19 of carrying out the Interaction

202.5.2.1 Interaction Based Information Alignment

21In some Choreographies there may be a requirement that, when the Interaction
22is performed, the Roles in the Choreography have agreement on the outcome.
23More specifically within an Interaction, a Role MAY need to have a common
24understanding of the observable information creations or changes of one or
25more *State Capturing Variables* that are complementary to one or more *State*
26*Capturing Variables* of its partner Role. Additionally, within an Interaction a Role
27MAY need to have a common understanding of the values of the Information
28Exchange Capturing Variables at the partner Role.

29For example, after an Interaction happens, both the Buyer and the Seller want to
30have a common understanding that:

- 31 • State Capturing Variables, such as "Order State", that contain
32 observable information at the Buyer and Seller, have values that are
33 complementary to each other, e.g. "Sent" at the Buyer and "Received"
34 at the Seller, and
- 35 • Information Exchange Capturing Variables have the same types with
36 the same content, e.g. The "Order" Variables at the Buyer and Seller
37 have the same Information Types and hold the same order information

38

39In WS-CDL an *alignment* Interaction MUST be explicitly used, in the cases
40where two interacting parties require the alignment of their observable

1 information changes and also the values of their exchanged information. After
2 the alignment Interaction completes, both parties progress at the same time, in a
3 lock-step fashion and the Variable information in both parties is aligned. Their
4 Variable alignment comes from the fact that the requesting party has to know
5 that the accepting party has received the message and the other way around,
6 the accepting party has to know that the requesting party has sent the message
7 before both of them progress. There is no intermediate state, where one party
8 sends a message and then it proceeds independently or the other party receives
9 a message and then it proceeds independently.

10 2.5.2.2 Interaction Life-line

11 An Interaction completes normally when the message exchange completes
12 successfully.

13 An Interaction completes abnormally when:

- 14 • An application signals an error condition during the management of a
15 request or within a party when processing the request
- 16 • The time-to-complete timeout, identifying the timeframe within which an
17 Interaction MUST complete, occurs after the Interaction was initiated
18 but before it completed
- 19 • Other types of errors, such as Protocol Based Exchange failures,
20 Security failures, Document Validation errors

21 2.5.2.3 Interaction Syntax

22 The syntax of the *interaction* construct is:

```
24 <interaction name="ncname"  
25     channelVariable="qname"  
26     operation="ncname"  
27     time-to-complete="xsd:duration"?  
28     align="true"|"false"?  
29     initiate="true"|"false"? >  
30  
31     <participate relationship="qname"  
32         fromRole="qname" toRole="qname" />  
33  
34     <exchange name="ncname"  
35         informationType="qname"? | channelType="qname"?  
36         action="request"|"respond" >  
37         <send variable="XPath-expression"?  
38             recordReference="list of ncname"?  
39             causeException="true"|"false"? />  
40         <receive variable="XPath-expression"?  
41             recordReference="list of ncname"?  
42             causeException="true"|"false"? />  
43     </exchange>  
44  
45     <record name="ncname"  
46         when="before"|"after"  
47         causeException="true"|"false"? >  
48
```

```
1 <source variable="XPath-expression"? | expression="Xpath-expression"? />  
2 <target variable="XPath-expression" />  
3 </record>*  
4 </interaction>
```

5The attribute name is used for specifying a name for each Interaction element
6declared within a Choreography.

7The channelVariable attribute specifies the Channel Variable containing information
8of a party, being the target of the Interaction, which is used for determining
9where and how to send and receive information to and into the party. The
10Channel Variable used in an Interaction MUST be available at the two Roles
11before the Interaction occurs.

12At runtime, information about a Channel Variable is expanded further. This
13requires that the messages in the Choreography also contain correlation
14information, for example by including:

- 15 • A protocol header, such as a SOAP header, that specifies the
16 correlation data to be used with the Channel, or
- 17 • Using the actual value of data within a message, for example the
18 "Order Number" of the Order that is common to all the messages sent
19 over the Channel

20In practice, when a Choreography is performed, several different ways of doing
21correlation may be employed which vary depending on the Channel Type.

22The operation attribute specifies the name of the operation that is associated with
23this Interaction. The specified operation belongs to the interface, as identified by
24the role and behavior elements of the Channel Type of the Channel Variable used
25in this Interaction.

26The optional time-to-complete attribute identifies the timeframe within which an
27Interaction MUST complete after it was initiated.

28The optional align attribute when set to "true" means that the Interaction results in
29the common understanding of both the information exchanged and the resulting
30observable information creations or changes at the ends of the Interaction as
31specified in the fromRole and the toRole. The default for this attribute is "false".

32An Interaction activity can be marked as a Choreography initiator when the
33optional initiate attribute is set to "true". The default for this attribute is "false".

34Within the participate element, the relationship attribute specifies the Relationship
35Type this Interaction participates in and the fromRole and toRole attributes specify
36the requesting and the accepting Role Types respectively. The Role Type
37identified by the toRole attribute MUST be the same as the Role Type identified
38by the role element of the Channel Type of the Channel Variable used in the
39interaction activity.

40The optional exchange element allows information to be exchanged during an
41Interaction. Within this element, the attribute name is used for specifying a name
42for it.

1 Within the `exchange` element, the optional attributes `informationType` and `channelType`
2 identify the Information Type or the Channel Type of the information that is
3 exchanged between the two Roles in an Interaction. If none of these attributes
4 are specified, then it is assumed that either no actual information is exchanged
5 or the type of information being exchanged is of no interest to the Choreography
6 definition.

7 Within the `exchange` element, the attribute `action` specifies the direction of the
8 information exchanged in the Interaction:

- 9 • When the `action` attribute is set to "request", then the information
10 exchange happens from `fromRole` to `toRole`
- 11 • When the `action` attribute is set to "respond", then the information
12 exchange happens from `toRole` to `fromRole`

13 Within the `exchange` element, the `send` element shows that information is sent from
14 a Role and the `receive` element shows that information is received at a Role
15 respectively in the Interaction:

- 16 • The `send` and the `receive` elements MUST only use the WS-CDL function
17 `getVariable` within the `variable` attribute
- 18 • The optional Variables specified within the `send` and `receive` elements
19 MUST be of type as described in the `informationType` OR `channelType` attributes
- 20 • When the `action` element is set to "request", then the Variable specified
21 within the `send` element using the `variable` attribute MUST be defined at the
22 `fromRole` and the Variable specified within the `receive` element using the
23 `variable` attribute MUST be defined at the `toRole`
- 24 • When the `action` element is set to "respond", then the Variable specified
25 within the `send` element using the `variable` attribute MUST be defined at the
26 `toRole` and the Variable specified within the `receive` element using the
27 `variable` attribute MUST be defined at `fromRole`
- 28 • The Variable specified within the `receive` element MUST not be defined
29 with the attribute `silent` set to "true"
- 30 • Within the `send` or the `receive` element(s) of an `exchange` element, the
31 `recordReference` attribute contains a list of references to `record` element(s) in
32 the same Interaction. The same `record` element MAY be referenced from
33 different `send` or the `receive` element(s) within the same Interaction thus
34 enabling re-use
- 35 • Within the `send` or the `receive` element(s) of an `exchange` element, the
36 `causeException` attribute when set to "true", specifies that an Exception will

- 1 be caused at the respective Roles. In this case, the informationType of the
2 exchange element MUST be of Exception Type
- 3 • The request exchange MUST NOT have causeException attribute set to
4 "true"
 - 5 • When two or more respond exchanges are specified, one respond
6 exchange MAY be of normal informationType and all others MUST be of
7 Exception Type. There is an implicit choice between two or more respond
8 exchanges
 - 9 • If the align attribute is set to "false" for the Interaction, then it means that
10 the:
 - 11 ○ Request exchange completes successfully for the requesting Role
12 once it has successfully sent the information of the Variable
13 specified within the send element and the Request exchange
14 completes successfully for the accepting Role once it has
15 successfully received the information of the Variable specified
16 within the receive element
 - 17 ○ Response exchange completes successfully for the accepting Role
18 once it has successfully sent the information of the Variable
19 specified within the send element and the Response exchange
20 completes successfully for the requesting Role once it has
21 successfully received the information of the Variable specified
22 within the receive element
 - 23 • If the align attribute is set to "true" for the Interaction, then it means that
24 the:
 - 25 ○ Interaction completes successfully if the Request and the
26 Response exchanges complete successfully and all referenced
27 records complete successfully
 - 28 ○ Request exchange completes successfully once both the
29 requesting Role has successfully sent the information of the
30 Variable specified within the send element and the accepting Role
31 has successfully received the information of the Variable specified
32 within the receive element
 - 33 ○ Response exchange completes successfully once both the
34 accepting Role has successfully sent the information of the
35 Variable specified within the send element and the requesting Role
36 has successfully received the information of the Variable specified
37 within the receive element

1The optional element `record` is used to create or change one or more Variables
2using another Variable or an expression. Within this element, the attribute `name`
3is used for specifying a name for it. Within the `record` element, the `source` and `target`
4elements specify these recordings of information at the send and receive ends of
5the Interaction:

- 6 • When the `action` element is set to "request", then the recording(s)
7 specified within the `source` and the `target` elements occur at the `fromRole`
8 for the send and at the `toRole` for the receive
- 9 • When the `action` element is set to "response", then the recording(s)
10 specified within the `source` and the `target` elements occur at the `toRole` for
11 the send and at the `fromRole` for the receive

12Within the `record` element, the `when` attribute specifies if a recording happens
13before or after a send or "before" or "after" a receive of a message at a Role in a
14Request or a Response exchange. If two or more `record` elements have the same
15value in their `when` attribute and are referenced within the `recordReference` attribute
16of a send or a receive element, then they are performed in the order in which they
17are specified.

18The following rules apply for the information recordings when using the `record`
19element:

- 20 • The `source` MUST define either a `variable` attribute or an `expression` attribute:
 - 21 ○ When the `source` defines an `expression` attribute this MUST contain
22 expressions, as defined in Section 2.4.3. The resulting type of the
23 defined expression MUST be compatible with the target Variable
24 type
 - 25 ○ When the `source` defines a Variable, then the `source` and the target
26 Variable MUST be of compatible type
 - 27 ○ When the `source` defines a Variable, then the `source` and the target
28 Variable MUST be defined at the same Role
- 29 • When the attribute `variable` is defined it MUST use only the WS-CDL
30 function `getVariable`
- 31 • The target Variable MUST NOT be defined with the attribute `silent` set to
32 "true"
- 33 • One or more `record` elements MAY be specified and performed at one or
34 both the Roles within an Interaction
- 35 • A `record` element MUST NOT be specified in the absence of an `exchange`
36 element
- 37 • The attribute `causeException` MAY be set to "true" in a `record` element if the
38 target Variable is an Exception Variable
- 39 • When the attribute `causeException` is set to "true" in a `record` element, the
40 corresponding Role gets into Exception state

- 1 • When two or more record elements are specified for the same Role in an
2 Interaction with target Variables of Exception Type, one of the Exception
3 recordings MAY occur. An Exception recording has an non-observable
4 predicate condition, associated implicitly with it, that decides if an
5 Exception occurs
- 6 • If the align attribute is set to "false" for the Interaction, then it means that
7 the Role specified within the record element makes available the creation
8 or change of the information specified within the record element
9 immediately after the successful completion of each record
- 10 • If the align attribute is set to "true" for the Interaction, then it means that
 - 11 ○ Both Roles know the availability of the creation or change of the
12 information specified within the record element only at the
13 successful completion of the Interaction
 - 14 ○ If there are two or more record elements specified within an
15 Interaction, then all record operations MUST complete successfully
16 for the Interact to complete successfully. Otherwise, none of the
17 Variables specified in the target attribute will be affected

18

19The example below shows a complete Choreography that involves one
20Interaction performed from Role Type "Consumer" to Role Type "Retailer" on the
21Channel "retailer-channel" as a request/response exchange:

- 22 • The message "purchaseOrder" is sent from the "Consumer" to the
23 "Retailer" as a request message
- 24 • The message "purchaseOrderAck" is sent from the "Retailer" to the
25 "Consumer" as a response message
- 26 • The Variable "consumer-channel" is made available at the "Retailer"
27 using the record element
- 28 • The Interaction happens on the "retailer-channel", which has a Token
29 Type "purchaseOrderID" used as an identity element of the channel.
30 This identity element is used to identify the business process of the
31 "Retailer"
- 32 • The request message "purchaseOrder" contains the identity of the
33 "Retailer" business process as specified in the tokenLocator for
34 "purchaseOrder" message
- 35 • The response message "purchaseOrderAck" contains the identity of the
36 "Consumer" business process as specified in the tokenLocator for
37 "purchaseOrderAck" message
- 38 • The "consumer-channel" is sent as a part of "purchaseOrder"
39 Interaction from the "Consumer" to the "Retailer" on "retailer-channel"
40 during the request. Here the record element makes available the

- 1 "Consumer-channel" at the "Retailer" Role. If the align attribute was set
 2 to "true" for this Interaction, then it also means that the "Consumer"
 3 knows that the "Retailer" now has the contact information of the
 4 "Consumer". In another example, the "Consumer" could set its Variable
 5 "OrderSent" to "true" and the "Retailer" would set its Variable
 6 "OrderReceived" to "true" using the record element
- 7 • The exchange "badPurchaseOrderAckException" specifies that an
 8 Exception of "badPOAckType" Exception Type could occur at both
 9 parties

10

```

11 <package name="ConsumerRetailerChoreography" version="1.0"
12   <informationType name="purchaseOrderType" type="pons:PurchaseOrderMsg"/>
13   <informationType name="purchaseOrderAckType" type="pons:PurchaseOrderAckMsg"/>
14   <informationType name="badPOAckType" type="xsd:string" exceptionType="true"/>
15
16   <token name="purchaseOrderID" informationType="tns:intType"/>
17   <token name="retailerRef" informationType="tns:uriType"/>
18   <tokenLocator tokenName="tns:purchaseOrderID"
19     informationType="tns:purchaseOrderType" query="/PO/orderId"/>
20   <tokenLocator tokenName="tns:purchaseOrderID"
21     informationType="tns:purchaseOrderAckType" query="/PO/orderId"/>
22
23   <roleType name="Consumer">
24     <behavior name="consumerForRetailer" interface="cns:ConsumerRetailerPT"/>
25     <behavior name="consumerForWarehouse" interface="cns:ConsumerWarehousePT"/>
26   </roleType>
27   <roleType name="Retailer">
28     <behavior name="retailerForConsumer" interface="rns:RetailerConsumerPT"/>
29   </roleType>
30
31   <relationshipType name="ConsumerRetailerRelationship">
32     <role type="tns:Consumer" behavior="consumerForRetailer"/>
33     <role type="tns:Retailer" behavior="retailerForConsumer"/>
34   </relationshipType>
35
36   <channelType name="ConsumerChannel">
37     <role type="tns:Consumer"/>
38     <reference>
39       <token type="tns:consumerRef"/>
40     </reference>
41     <identity>
42       <token type="tns:purchaseOrderID"/>
43     </identity>
44   </channelType>
45
46   <channelType name="RetailerChannel">
47     <passing channel="ConsumerChannel" action="request" />
48     <role type="tns:Retailer" behavior="retailerForConsumer"/>
49     <reference>
50       <token type="tns:retailerRef"/>
51     </reference>
52     <identity>
53       <token type="tns:purchaseOrderID"/>
54     </identity>
55   </channelType>
56
57   <choreography name="ConsumerRetailerChoreography" root="true">

```

```

1   <relationship type="tns:ConsumerRetailerRelationship"/>
2   <variableDefinitions>
3     <variable name="purchaseOrder" informationType="tns:purchaseOrderType"
4       silent="true" />
5     <variable name="purchaseOrderAck"
6       informationType="tns:purchaseOrderAckType" />
7     <variable name="retailer-channel" channelType="tns:RetailerChannel"/>
8     <variable name="consumer-channel" channelType="tns:ConsumerChannel"/>
9     <variable name="badPurchaseOrderAck"
10      informationType="tns:badPOAckType" role="tns:Consumer"/>
11    <variable name="badPurchaseOrderAck"
12      informationType="tns:badPOAckType" role="tns:Retailer"
13      silent="true" />
14  </variableDefinitions>
15
16  <interaction name="createPO"
17    channelVariable="tns:retailer-channel"
18    operation="handlePurchaseOrder" align="true"
19    initiate="true">
20    <participate relationship="tns:ConsumerRetailerRelationship"
21      fromRole="tns:Consumer" toRole="tns:Retailer"/>
22
23    <exchange name="request"
24      informationType="tns:purchaseOrderType" action="request">
25      <send variable="cdl:getVariable("tns:purchaseOrder", "", "")" />
26      <receive variable="cdl:getVariable("tns:purchaseOrder", "", "")"
27        recordReference="record-the-channel-info" />
28    </exchange>
29
30    <exchange name="response"
31      informationType="purchaseOrderAckType" action="respond">
32      <send variable="cdl:getVariable("tns:purchaseOrderAck", "", "")" />
33      <receive variable="cdl:getVariable("tns:purchaseOrderAck", "", "")" />
34      recordReference=" recordBadPurchaseOrder " />
35    </exchange>
36
37    <exchange name="badPurchaseOrderAckException"
38      informationType="badPOAckType" action="respond">
39      <send variable="cdl:getVariable("tns:badPurchaseOrderAck", "", "")"
40        causeException="true" />
41      <receive variable="cdl:getVariable("tns:badPurchaseOrderAck", "", "")"
42        causeException="true" />
43    </exchange>
44
45    <record name="record-the-channel-info" when="after">
46      <source variable="cdl:getVariable("tns:purchaseOrder, "",
47        "PO/CustomerRef")"/>
48      <target variable="cdl:getVariable("tns:consumer-channel", "", "")"/>
49    </record>
50
51  </interaction>
52 </choreography>
53 </package>

```

12.5.3 Composing Choreographies

2The *perform* activity realizes the “composition of Choreographies”, whereas
3combining existing Choreographies results in the creation of new
4Choreographies. For example if two separate Choreographies were defined as
5follows:

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

13You could then create a new "Quote and Order" Choreography by reusing the
14two where the RFQ Choreography was performed first, and then, depending on
15the outcome of the RFQ Choreography, the order was placed using the Order
16Placement Choreography. In this case the new Choreography is "composed" out
17of the two previously defined Choreographies. Using this approach,
18Choreographies can be combined to support Choreographies of any required
19complexity allowing more flexibility as Choreographies defined elsewhere can be
20reused.

21The *perform* activity enables a Choreography to specify that another
22Choreography is performed at this point in its definition, as an enclosed
23Choreography. The performed Choreography even when defined in a different
24Choreography Package is conceptually treated as an enclosed Choreography.
25An enclosing Choreography MAY perform only an immediately contained
26Choreography that is Locally defined.

27

28The syntax of the *perform* construct is:

```
30 <perform choreographyName="qname">  
31   <bind name="ncname">  
32     <this variable="XPath-expression" role="qname"/>  
33     <free variable="XPath-expression" role="qname"/>  
34   </bind>  
35   choreography?  
36 </perform>
```

38Within the *perform* element the *choreographyName* attribute references a Locally or
39Globally defined Choreography to be performed.

40The optional *choreography* within the *perform* element defines a Locally defined
41Choreography that is performed only by this *perform* activity. If specified the
42*choreographyName* attribute within the *perform* element MUST match the attribute
43*name* within the *choreography* element of the *choreography*.

1The optional bind element within the perform element enables information in the
2performing Choreography to be shared with the performed Choreography and
3vice versa. Within the bind element, the attribute name is used for specifying a
4name for each bind element declared within this perform activity. Within the bind
5element, the role attribute aliases the Roles from the performing Choreography to
6the performed Choreography.

7The variable attribute within this element specifies that a Variable in the performing
8Choreography is bound with the Variable identified by variable attribute within the
9free element in the performed Choreography.

10The following rules apply when a Choreography is performed:

- 11 • The Choreography to be performed MUST NOT be a root Choreography
- 12 • The Choreography to be performed MUST be defined either using a
13 choreography immediately contained in the same Choreography or it
14 MUST be a top-level Choreography with root attribute set to "false" in the
15 same or different Choreography Package. Performed Choreographies
16 that are declared in a different Choreography Package MUST be included
17 first before they can be performed
- 18 • The Role Types within a single bind element MUST be carried out by the
19 same party, hence they MUST belong to the same Participant Type
- 20 • The variable attribute within this element and free element MUST define only
21 the WS-CDL function `getVariable`
- 22 • The free Variables within the bind element MUST have the attribute `free` set
23 to "true" in their definition
- 24 • There MUST not be a cyclic dependency on the Choreographies
25 performed. For example Choreography C1 is performing Choreography
26 C2 which is performing Choreography C1 again

27

28The example below shows a Choreography composition, where a Choreography
29"PurchaseChoreography" is performing the Globally defined Choreography
30"RetailerWarehouseChoreography" and aliases the Variable
31"purchaseOrderAtRetailer" to the Variable "purchaseOrder" defined at the
32performed Choreography "RetailerWarehouseChoreography". Once aliased, the
33Variable "purchaseOrderAtRetailer" extends to the enclosed Choreography and
34thus these Variables can be used interchangeably for sharing their information.

```
36 <choreography name="PurchaseChoreography">  
37   ...  
38   <variableDefinitions>  
39     <variable name="purchaseOrderAtRetailer"  
40       informationType="purchaseOrder" role="tns:Retailer"/>  
41   </variableDefinitions>  
42   ...  
43   <perform choreographyName="RetailerWarehouseChoreography">
```

```

1      <bind name="aliasRetailer">
2          <this variable="cdl:getVariable("tns:purchaseOrderAtRetailer", "", "")"
3              role="tns:Retailer"/>
4          <free variable="cdl:getVariable("tns:purchaseOrder", "", "")"
5              role="tns:Retailer"/>
6      </bind>
7  </perform>
8      ...
9 </choreography>
10
11 <choreography name="RetailerWarehouseChoreography">
12     <variableDefinitions>
13         <variable name="purchaseOrder"
14             informationType="purchaseOrder" role="tns:Retailer" free="true"/>
15     </variableDefinitions>
16     ...
17 </choreography>

```

182.5.4 Assigning Variables

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

22

23 The syntax of the *assign* construct is:

```

25 <assign role="qname">
26     <copy name="ncname">
27         <source variable="XPath-expression"?|expression="XPath-expression"? />
28         <target variable="XPath-expression" />
29     </copy>+
30 </assign>

```

31 A copy element within the assign element creates or changes at a Role the
32 Variable defined by the target element using the Variable or expression defined
33 by the source element at the same Role. Within the copy element, the attribute
34 name is used for specifying a name for each copy element declared within this
35 assign activity.

36 The following rules apply to assignment:

- 37
- The source MUST define either a variable attribute or an expression attribute:
 - 38 ○ When the source defines an expression attribute this MUST contain
39 expressions, as defined in Section 2.4.3. The resulting type of the
40 defined expression MUST be compatible with the target Variable
41 type
 - 42 ○ When the source defines a Variable, then the source and the target
43 Variable MUST be of compatible type
 - 44 ○ When the source defines a Variable, then the source and the target
45 Variable MUST be defined at the same Role

49

- 1 • When the attribute `variable` is defined it MUST use only the WS-CDL
2 function `getVariable`
- 3 • The target Variable MUST NOT be defined with the attribute `silent` set to
4 “true”
- 5 • When two or more `copy` elements belong to the same `assign` element, then
6 they are performed in the order in which they are defined.
- 7 • If there are two or more `copy` elements specified within an `assign`, then all
8 copy operations MUST complete successfully for the `assign` to complete
9 successfully. Otherwise, none of the Variables specified in the target
10 attribute will be affected

11

12 The examples below show some possible usages of `assign`.

14 **Example 1:**

```
16 <assign role="tns:Retailer">
17   <copy name="copyAddressInfo">
18     <source variable="cdl:getVariable("PurchaseOrderMsg", "",
19                                     "/PO/CustomerAddress)" />
20     <target variable="cdl:getVariable("CustomerAddress", "", "")" />
21   </copy>
22 </assign>
```

23

24 **Example 2:**

```
26 <assign role="tns:Retailer">
27   <copy name="copyPriceInfo">
28     <source expression="(10+237)/34" />
29     <target variable="cdl:getVariable("ProductPrice", "", "", "tns:Retailer)" />
30   </copy>
31 </assign>
```

32

33 **Example 3:**

```
34 <assign role="tns:Customer">
35   <copy name="copyLiteral">
36     <source expression="Hello World" />
37     <target variable="cdl:getVariable("VarName", "", "", "tns:Customer)" />
38   </copy>
39 </assign>
```

43 2.5.5 Marking Silent Actions

44 *Silent actions* are explicit designators used for marking the points where party
45 specific operations with non-observable operational details MUST be performed.

46 For example, the mechanism for checking the inventory of a warehouse should
47 not be observable to other parties but the fact that the inventory level does

1influence the global observable behavior with a buyer party needs to be specified
2in the Choreography definition.

3The syntax of the *silent action* construct is:

```
5 <silentAction role="qname? />
```

6The optional attribute `role` is used to specify the party at which the silent action
7will be performed. If a silent action is defined without a Role, it is implied that the
8action is performed at all the Roles that are part of the Relationships of the
9Choreography this activity is enclosed within.

102.5.6 Marking the Absence of Actions

11*No actions* are explicit designators used for marking the points where a party
12does not perform any action.

13The syntax of the *no action* construct is:

```
15 <noAction role="qname? />
```

16The optional attribute `role` is used to specify the party at which no action will be
17performed. If a `noAction` is defined without a Role, it is implied that no action will
18be performed at any of the Roles that are part of the Relationships of the
19Choreography this activity is enclosed within.

203 Example

21To be completed

224 Relationship with the Security framework

23Because messages can have consequences in the real world, the collaboration
24parties will impose security requirements on the information exchanges. Many of
25these requirements can be satisfied by the use of WS-Security [24].

265 Relationship with the Reliable Messaging 27 framework

28The WS-Reliability specification [22] provides a reliable mechanism to exchange
29information among collaborating parties. The WS-Reliability specification
30prescribes the formats for all information exchanged without placing any
31restrictions on the content of the encapsulated business documents. The WS-
32Reliability specification supports message exchange patterns, over various

1transport protocols (examples are HTTP/S, FTP, SMTP, etc.). The WS-Reliability
2specification supports sequencing of messages and guaranteed, exactly once
3delivery.

4A violation of any of these consistency guarantees results in an error condition,
5which MAY be reflected in the Choreography with an Exception.

6 Relationship with the Transaction/Coordination 7 framework

8In WS-CDL, two parties make progress by interacting. In the cases where two
9interacting parties require the alignment of their Variables capturing observable
10information changes or their exchanged information between them, an alignment
11Interaction is modeled in a Choreography. After the alignment Interaction
12completes, both parties progress at the same time, in a lock-step fashion. The
13Variable information alignment comes from the fact that the requesting party has
14to know that the accepting party has received the message and the other way
15around, the accepting party has to know that the requesting party has sent the
16message before both of them progress. There is no intermediate state, where
17one party sends a message and then it proceeds independently or the other
18party receives a message and then it proceeds independently.

19Implementing this type of handshaking in a distributed system requires support
20from a Transaction/Coordination protocol, where agreement of the outcome
21among parties can be reached even in the case of failures and loss of
22messages.

23 7 Acknowledgments

24To be completed

25 8 References

26[1] S. Bradner, "Key words for use in RFCs to Indicate Requirement Levels", RFC 2119, Harvard
27University, March 1997

28[2] T. Berners-Lee, R. Fielding, L. Masinter, "Uniform Resource Identifiers (URI): Generic Syntax",
29RFC 2396, MIT/LCS, U.C. Irvine, Xerox Corporation, August 1998.

30[3] <http://www.w3.org/TR/html401/interaction/forms.html#submit-format>

31[4] <http://www.w3.org/TR/html401/appendix/notes.html#ampersands-in-uris>

32[5] <http://www.w3.org/TR/html401/interaction/forms.html#h-17.13.4>

33[6] Simple Object Access Protocol (SOAP) 1.1 "http://www.w3.org/TR/2000/NOTE-SOAP-
3420000508/"

35[7] Web Services Definition Language (WSDL) 2.0

36[8] Industry Initiative "Universal Description, Discovery and Integration"

- 1[9] W3C Recommendation "The XML Specification"
- 2[10] XML-Namespaces "Namespaces in XML, Tim Bray et al., eds., W3C, January 1999"
- 3<http://www.w3.org/TR/REC-xml-names>
- 4[11] W3C Working Draft "XML Schema Part 1: Structures". This is work in progress.
- 5[12] W3C Working Draft "XML Schema Part 2: Datatypes". This is work in progress.
- 6[13] W3C Recommendation "XML Path Language (XPath) Version 1.0"
- 7[14] "Uniform Resource Identifiers (URI): Generic Syntax," RFC 2396, T. Berners-Lee, R. Fielding, 8L. Masinter, MIT/LCS, U.C. Irvine, Xerox Corporation, August 1998.
- 9[15] WSCI: Web Services Choreography Interface 1.0, A. Arkin et.al
- 10[16] XLANG: Web Services for Business Process Design, S. Thatte, 2001 Microsoft Corporation
- 11[17] WSFL: Web Service Flow Language 1.0, F. Leymann, 2001 IBM Corporation
- 12[18] OASIS Working Draft "BPEL: Business Process Execution Language 2.0". This is work in
13progress.
- 14[19] BPMI.org "BPML: Business Process Modeling Language 1.0"
- 15[20] Workflow Management Coalition "XPDL: XML Processing Description Language 1.0", M.
16Marin, R. Norin R. Shapiro
- 17[21] OASIS Working Draft "WS-CAF: Web Services Context, Coordination and Transaction
18Framework 1.0". This is work in progress.
- 19[22] OASIS Working Draft "Web Services Reliability 1.0". This is work in progress.
- 20[23] The Java Language Specification
- 21[24] OASIS "Web Services Security"
- 22[25] J2EE: Java 2 Platform, Enterprise Edition, Sun Microsystems
- 23[26] ECMA. 2001. Standard ECMA-334: C# Language Specification
- 24[27] "XML Inclusions Version 1.0" <http://www.w3.org/TR/xinclude/>

259 WS-CDL XSD Schemas

```

26 <?xml version="1.0" encoding="UTF-8"?>
27 <schema
28     targetNamespace="http://www.w3.org/ws/choreography/2004/09/WSCDL/"
29     xmlns="http://www.w3.org/2001/XMLSchema"
30     xmlns:cdl="http://www.w3.org/ws/choreography/2004/09/WSCDL/"
31     elementFormDefault="qualified">
32
33     <complexType name="tExtensibleElements">
34         <annotation>
35             <documentation>
36                 This type is extended by other CDL component types to allow
37                 elements and attributes from other namespaces to be added.
38                 This type also contains the optional description element that
39                 is applied to all CDL constructs.
40             </documentation>
41         </annotation>
42         <sequence>
43             <element name="description" minOccurs="0">
44                 <complexType mixed="true">

```

```

1      <sequence minOccurs="0" maxOccurs="unbounded">
2          <any processContents="lax"/>
3      </sequence>
4      <attribute name="type" type="cdl:tDescriptionType" use="optional"
5          default="documentation"/>
6      </complexType>
7  </element>
8      <any namespace="##other" processContents="lax"
9          minOccurs="0" maxOccurs="unbounded"/>
10 </sequence>
11 <anyAttribute namespace="##other" processContents="lax"/>
12
13 </complexType>
14
15 <element name="package" type="cdl:tPackage"/>
16
17 <complexType name="tPackage">
18     <complexContent>
19         <extension base="cdl:tExtensibleElements">
20             <sequence>
21                 <element name="informationType" type="cdl:tInformationType"
22                     minOccurs="0" maxOccurs="unbounded"/>
23                 <element name="token" type="cdl:tToken" minOccurs="0"
24                     maxOccurs="unbounded"/>
25                 <element name="tokenLocator" type="cdl:tTokenLocator"
26                     minOccurs="0" maxOccurs="unbounded"/>
27                 <element name="roleType" type="cdl:tRoleType" minOccurs="0"
28                     maxOccurs="unbounded"/>
29                 <element name="relationshipType" type="cdl:tRelationshipType"
30                     minOccurs="0" maxOccurs="unbounded"/>
31                 <element name="participantType" type="cdl:tParticipantType"
32                     minOccurs="0" maxOccurs="unbounded"/>
33                 <element name="channelType" type="cdl:tChannelType"
34                     minOccurs="0" maxOccurs="unbounded"/>
35                 <element name="choreography" type="cdl:tChoreography"
36                     minOccurs="0" maxOccurs="unbounded"/>
37             </sequence>
38             <attribute name="name" type="NCName" use="required"/>
39             <attribute name="author" type="string" use="optional"/>
40             <attribute name="version" type="string" use="required"/>
41             <attribute name="targetNamespace" type="anyURI"
42                 use="required"/>
43         </extension>
44     </complexContent>
45 </complexType>
46
47 <complexType name="tInformationType">
48     <complexContent>
49         <extension base="cdl:tExtensibleElements">
50             <attribute name="name" type="NCName" use="required"/>
51             <attribute name="type" type="QName" use="optional"/>
52             <attribute name="element" type="QName" use="optional"/>
53             <attribute name="exceptionType" type="boolean" use="optional"
54                 default="false" />
55         </extension>
56     </complexContent>
57 </complexType>
58
59 <complexType name="tToken">
60     <complexContent>
61         <extension base="cdl:tExtensibleElements">

```

```

1      <attribute name="name" type="NCName" use="required"/>
2      <attribute name="informationType" type="QName"
3          use="required"/>
4      </extension>
5  </complexContent>
6 </complexType>
7
8 <complexType name="tTokenLocator">
9   <complexContent>
10    <extension base="cdl:tExtensibleElements">
11      <attribute name="tokenName" type="QName" use="required"/>
12      <attribute name="informationType" type="QName"
13          use="required"/>
14      <attribute name="part" type="NCName" use="optional" />
15      <attribute name="query" type="cdl:tXPath-expr"
16          use="required"/>
17    </extension>
18  </complexContent>
19 </complexType>
20
21 <complexType name="tRoleType">
22   <complexContent>
23    <extension base="cdl:tExtensibleElements">
24      <sequence>
25        <element name="behavior" type="cdl:tBehavior"
26            maxOccurs="unbounded"/>
27      </sequence>
28      <attribute name="name" type="NCName" use="required"/>
29    </extension>
30  </complexContent>
31 </complexType>
32
33 <complexType name="tBehavior">
34   <complexContent>
35    <extension base="cdl:tExtensibleElements">
36      <attribute name="name" type="NCName" use="required"/>
37      <attribute name="interface" type="QName" use="optional"/>
38    </extension>
39  </complexContent>
40 </complexType>
41
42 <complexType name="tRelationshipType">
43   <complexContent>
44    <extension base="cdl:tExtensibleElements">
45      <sequence>
46        <element name="role" type="cdl:tRoleRef" minOccurs="2"
47            maxOccurs="2"/>
48      </sequence>
49      <attribute name="name" type="NCName" use="required"/>
50    </extension>
51  </complexContent>
52 </complexType>
53
54 <complexType name="tRoleRef">
55   <complexContent>
56    <extension base="cdl:tExtensibleElements">
57      <attribute name="type" type="QName" use="required"/>
58      <attribute name="behavior" use="optional">
59        <simpleType>
60          <list itemType="NCName"/>
61        </simpleType>

```

```

1      </attribute>
2      </extension>
3      </complexContent>
4  </complexType>
5
6  <complexType name="tParticipantType">
7      <complexContent>
8          <extension base="cdl:tExtensibleElements">
9              <sequence>
10                 <element name="role" type="cdl:tRoleRef2"
11                     maxOccurs="unbounded"/>
12             </sequence>
13             <attribute name="name" type="NCName" use="required"/>
14         </extension>
15     </complexContent>
16 </complexType>
17
18 <complexType name="tRoleRef2">
19     <complexContent>
20         <extension base="cdl:tExtensibleElements">
21             <attribute name="type" type="QName" use="required"/>
22         </extension>
23     </complexContent>
24 </complexType>
25
26 <complexType name="tChannelType">
27     <complexContent>
28         <extension base="cdl:tExtensibleElements">
29             <sequence>
30                 <element name="passing" type="cdl:tPassing" minOccurs="0"
31                     maxOccurs="unbounded"/>
32                 <element name="role" type="cdl:tRoleRef3"/>
33                 <element name="reference" type="cdl:tReference"/>
34                 <element name="identity" type="cdl:tIdentity" minOccurs="0"
35                     maxOccurs="1"/>
36             </sequence>
37             <attribute name="name" type="NCName" use="required"/>
38             <attribute name="usage" type="cdl:tUsage" use="optional"
39                 default="unlimited"/>
40             <attribute name="action" type="cdl:tAction" use="optional"
41                 default="request-respond"/>
42         </extension>
43     </complexContent>
44 </complexType>
45
46 <complexType name="tRoleRef3">
47     <complexContent>
48         <extension base="cdl:tExtensibleElements">
49             <attribute name="type" type="QName" use="required"/>
50             <attribute name="behavior" type="NCName" use="optional"/>
51         </extension>
52     </complexContent>
53 </complexType>
54
55 <complexType name="tPassing">
56     <complexContent>
57         <extension base="cdl:tExtensibleElements">
58             <attribute name="channel" type="QName" use="required"/>
59             <attribute name="action" type="cdl:tAction" use="optional"
60                 default="request-respond"/>
61             <attribute name="new" type="boolean" use="optional"

```

```

1         default="true"/>
2     </extension>
3 </complexContent>
4 </complexType>
5
6 <complexType name="tReference">
7     <complexContent>
8         <extension base="cdl:tExtensibleElements">
9             <sequence>
10                <element name="token" type="cdl:tTokenReference"
11                    minOccurs="1" maxOccurs="1"/>
12            </sequence>
13        </extension>
14    </complexContent>
15 </complexType>
16
17 <complexType name="tTokenReference">
18     <complexContent>
19         <extension base="cdl:tExtensibleElements">
20             <attribute name="name" type="QName" use="required"/>
21         </extension>
22     </complexContent>
23 </complexType>
24
25 <complexType name="tIdentity">
26     <complexContent>
27         <extension base="cdl:tExtensibleElements">
28             <sequence>
29                <element name="token" type="cdl:tTokenReference"
30                    minOccurs="1" maxOccurs="unbounded"/>
31            </sequence>
32        </extension>
33    </complexContent>
34 </complexType>
35
36
37 <complexType name="tChoreography">
38     <complexContent>
39         <extension base="cdl:tExtensibleElements">
40             <sequence>
41                <element name="relationship" type="cdl:tRelationshipRef"
42                    maxOccurs="unbounded"/>
43                <element name="variableDefinitions"
44                    type="cdl:tVariableDefinitions" minOccurs="0"/>
45                <element name="choreography" type="cdl:tChoreography"
46                    minOccurs="0" maxOccurs="unbounded"/>
47                <group ref="cdl:activity"/>
48                <element name="exception" type="cdl:tException"
49                    minOccurs="0"/>
50                <element name="finalizer" type="cdl:tFinalizer"
51                    minOccurs="0"/>
52            </sequence>
53            <attribute name="name" type="NCName" use="required"/>
54            <attribute name="complete" type="cdl:tBoolean-expr"
55                use="optional"/>
56            <attribute name="isolation" type="cdl:tIsolation"
57                use="optional" default="dirty-write"/>
58            <attribute name="root" type="boolean" use="optional"
59                default="false"/>
60        </extension>
61    </complexContent>

```

```

1   </complexType>
2
3   <complexType name="tRelationshipRef">
4     <complexContent>
5       <extension base="cdl:tExtensibleElements">
6         <attribute name="type" type="QName" use="required"/>
7       </extension>
8     </complexContent>
9   </complexType>
10
11  <complexType name="tVariableDefinitions">
12    <complexContent>
13      <extension base="cdl:tExtensibleElements">
14        <sequence>
15          <element name="variable" type="cdl:tVariable"
16            maxOccurs="unbounded"/>
17        </sequence>
18      </extension>
19    </complexContent>
20  </complexType>
21
22  <complexType name="tVariable">
23    <complexContent>
24      <extension base="cdl:tExtensibleElements">
25        <attribute name="name" type="NCName" use="required"/>
26        <attribute name="informationType" type="QName"
27          use="optional"/>
28        <attribute name="channelType" type="QName" use="optional"/>
29        <attribute name="mutable" type="boolean" use="optional"
30          default="true"/>
31        <attribute name="free" type="boolean" use="optional"
32          default="false"/>
33        <attribute name="silent" type="boolean" use="optional"
34          default="false"/>
35        <attribute name="role" type="QName" use="optional"/>
36      </extension>
37    </complexContent>
38  </complexType>
39
40  <group name="activity">
41    <choice>
42      <element name="sequence" type="cdl:tSequence"/>
43      <element name="parallel" type="cdl:tParallel"/>
44      <element name="choice" type="cdl:tChoice"/>
45      <element name="workunit" type="cdl:tWorkunit"/>
46      <element name="interaction" type="cdl:tInteraction"/>
47      <element name="perform" type="cdl:tPerform"/>
48      <element name="assign" type="cdl:tAssign"/>
49      <element name="silentAction" type="cdl:tSilentAction"/>
50      <element name="noAction" type="cdl:tNoAction"/>
51
52    </choice>
53  </group>
54
55  <complexType name="tSequence">
56    <complexContent>
57      <extension base="cdl:tExtensibleElements">
58        <sequence>
59          <group ref="cdl:activity" maxOccurs="unbounded"/>
60        </sequence>
61      </extension>

```

```

1   </complexContent>
2   </complexType>
3
4   <complexType name="tParallel">
5     <complexContent>
6       <extension base="cdl:tExtensibleElements">
7         <sequence>
8           <group ref="cdl:activity" maxOccurs="unbounded"/>
9         </sequence>
10      </extension>
11    </complexContent>
12  </complexType>
13  <complexType name="tChoice">
14    <complexContent>
15      <extension base="cdl:tExtensibleElements">
16        <sequence>
17          <group ref="cdl:activity" maxOccurs="unbounded"/>
18        </sequence>
19      </extension>
20    </complexContent>
21  </complexType>
22
23  <complexType name="tWorkunit">
24    <complexContent>
25      <extension base="cdl:tExtensibleElements">
26        <sequence>
27          <group ref="cdl:activity"/>
28        </sequence>
29        <attribute name="name" type="NCName" use="required"/>
30        <attribute name="guard" type="cdl:tBoolean-expr"
31          use="optional"/>
32        <attribute name="repeat" type="cdl:tBoolean-expr"
33          use="optional"/>
34        <attribute name="block" type="boolean"
35          use="optional" default="false"/>
36      </extension>
37    </complexContent>
38  </complexType>
39
40  <complexType name="tPerform">
41    <complexContent>
42      <extension base="cdl:tExtensibleElements">
43        <sequence>
44          <element name="bind" type="cdl:tBind"
45            minOccurs="0" maxOccurs="unbounded"/>
46          <element name="choreography" type="cdl:tChoreography"
47            minOccurs="0" maxOccurs="1"/>
48        </sequence>
49        <attribute name="choreographyName" type="QName"
50          use="required"/>
51      </extension>
52    </complexContent>
53  </complexType>
54
55  <complexType name="tBind">
56    <complexContent>
57      <extension base="cdl:tExtensibleElements">
58        <sequence>
59          <element name="this" type="cdl:tBindVariable"/>
60          <element name="free" type="cdl:tBindVariable"/>
61        </sequence>

```

```

1     </extension>
2     </complexContent>
3 </complexType>
4
5 <complexType name="tBindVariable">
6     <complexContent>
7         <extension base="cdl:tExtensibleElements">
8             <attribute name="variable" type="cdl:tXPath-expr"
9                 use="required"/>
10            <attribute name="role" type="QName" use="required"/>
11        </extension>
12    </complexContent>
13 </complexType>
14
15 <complexType name="tInteraction">
16     <complexContent>
17         <extension base="cdl:tExtensibleElements">
18             <sequence>
19                 <element name="participate" type="cdl:tParticipate"/>
20                 <element name="exchange" type="cdl:tExchange" minOccurs="0"
21                     maxOccurs="unbounded"/>
22                 <element name="record" type="cdl:tRecord" minOccurs="0"
23                     maxOccurs="unbounded"/>
24             </sequence>
25             <attribute name="name" type="NCName" use="required"/>
26             <attribute name="channelVariable" type="QName"
27                 use="required"/>
28             <attribute name="operation" type="NCName" use="required"/>
29             <attribute name="time-to-complete" type="duration"
30                 use="optional"/>
31             <attribute name="align" type="boolean" use="optional"
32                 default="false"/>
33             <attribute name="initiate" type="boolean"
34                 use="optional" default="false"/>
35         </extension>
36     </complexContent>
37 </complexType>
38
39 <complexType name="tParticipate">
40     <complexContent>
41         <extension base="cdl:tExtensibleElements">
42             <attribute name="relationship" type="QName" use="required"/>
43             <attribute name="fromRole" type="QName" use="required"/>
44             <attribute name="toRole" type="QName" use="required"/>
45         </extension>
46     </complexContent>
47 </complexType>
48
49 <complexType name="tExchange">
50     <complexContent>
51         <extension base="cdl:tExtensibleElements">
52             <sequence>
53                 <element name="send" type="cdl:tVariableRecordRef"/>
54                 <element name="receive" type="cdl:tVariableRecordRef"/>
55             </sequence>
56             <attribute name="name" type="string" use="required"/>
57             <attribute name="informationType" type="QName"
58                 use="optional"/>
59             <attribute name="channelType" type="QName"
60                 use="optional"/>
61             <attribute name="action" type="cdl:tAction2" use="required"/>

```

```

1     </extension>
2     </complexContent>
3 </complexType>
4
5 <complexType name="tVariableRecordRef">
6     <complexContent>
7         <extension base="cdl:tExtensibleElements">
8             <attribute name="variable" type="cdl:tXPath-expr"
9                 use="optional"/>
10            <attribute name="recordReference" use="optional">
11                <simpleType>
12                    <list itemType="NCName"/>
13                </simpleType>
14            </attribute>
15            <attribute name="causeException" type="boolean"
16                use="optional"/>
17        </extension>
18    </complexContent>
19 </complexType>
20
21 <complexType name="tSourceVariableRef">
22     <complexContent>
23         <extension base="cdl:tExtensibleElements">
24             <attribute name="variable" type="cdl:tXPath-expr"
25                 use="optional"/>
26             <attribute name="expression" type="cdl:tXPath-expr"
27                 use="optional"/>
28         </extension>
29    </complexContent>
30 </complexType>
31
32 <complexType name="tVariableRef">
33     <complexContent>
34         <extension base="cdl:tExtensibleElements">
35             <attribute name="variable" type="cdl:tXPath-expr"
36                 use="required"/>
37         </extension>
38    </complexContent>
39 </complexType>
40
41 <complexType name="tRecord">
42     <complexContent>
43         <extension base="cdl:tExtensibleElements">
44             <sequence>
45                 <element name="source" type="cdl:tSourceVariableRef"/>
46                 <element name="target" type="cdl:tVariableRef"/>
47             </sequence>
48             <attribute name="name" type="string" use="required"/>
49             <attribute name="causeException" type="boolean" use="optional"
50 default="false"/>
51             <attribute name="when" type="string" use="required"/>
52         </extension>
53    </complexContent>
54 </complexType>
55
56 <complexType name="tAssign">
57     <complexContent>
58         <extension base="cdl:tExtensibleElements">
59             <sequence>
60                 <element name="copy" type="cdl:tCopy"

```

```

1         maxOccurs="unbounded"/>
2     </sequence>
3     <attribute name="role" type="QName" use="required"/>
4 </extension>
5 </complexContent>
6 </complexType>
7
8 <complexType name="tCopy">
9     <complexContent>
10        <extension base="cdl:tExtensibleElements">
11            <sequence>
12                <element name="source" type="cdl:tSourceVariableRef"/>
13                <element name="target" type="cdl:tVariableRef"/>
14            </sequence>
15            <attribute name="name" type="NCName" use="required"/>
16        </extension>
17    </complexContent>
18 </complexType>
19
20 <complexType name="tSilentAction">
21     <complexContent>
22        <extension base="cdl:tExtensibleElements">
23            <attribute name="role" type="QName" use="optional"/>
24        </extension>
25    </complexContent>
26 </complexType>
27
28 <complexType name="tNoAction">
29     <complexContent>
30        <extension base="cdl:tExtensibleElements">
31            <attribute name="role" type="QName" use="optional"/>
32        </extension>
33    </complexContent>
34 </complexType>
35
36 <complexType name="tException">
37     <complexContent>
38        <extension base="cdl:tExtensibleElements">
39            <sequence>
40                <element name="workunit" type="cdl:tWorkunit"
41                    maxOccurs="unbounded"/>
42            </sequence>
43            <attribute name="name" type="NCName" use="required"/>
44        </extension>
45    </complexContent>
46 </complexType>
47
48 <complexType name="tFinalizer">
49     <complexContent>
50        <extension base="cdl:tExtensibleElements">
51            <sequence>
52                <element name="workunit" type="cdl:tWorkunit"/>
53            </sequence>
54            <attribute name="name" type="NCName" use="required"/>
55        </extension>
56    </complexContent>
57 </complexType>
58
59 <simpleType name="tAction">
60     <restriction base="string">
61         <enumeration value="request-respond"/>

```

```

1      <enumeration value="request"/>
2      <enumeration value="respond"/>
3    </restriction>
4  </simpleType>
5
6  <simpleType name="tAction2">
7    <restriction base="string">
8      <enumeration value="request"/>
9      <enumeration value="respond"/>
10   </restriction>
11 </simpleType>
12
13 <simpleType name="tUsage">
14   <restriction base="string">
15     <enumeration value="once"/>
16     <enumeration value="unlimited"/>
17   </restriction>
18 </simpleType>
19
20 <simpleType name="tBoolean-expr">
21   <restriction base="string"/>
22 </simpleType>
23
24 <simpleType name="tXPath-expr">
25   <restriction base="string"/>
26 </simpleType>
27
28 <simpleType name="tIsolation">
29   <restriction base="string">
30     <enumeration value="dirty-write"/>
31     <enumeration value="dirty-read"/>
32     <enumeration value="serializable"/>
33   </restriction>
34 </simpleType>
35
36 <simpleType name="tDescriptionType">
37   <restriction base="string">
38     <enumeration value="documentation"/>
39     <enumeration value="reference"/>
40     <enumeration value="semantics"/>
41   </restriction>
42 </simpleType>
43
44 </schema>

```