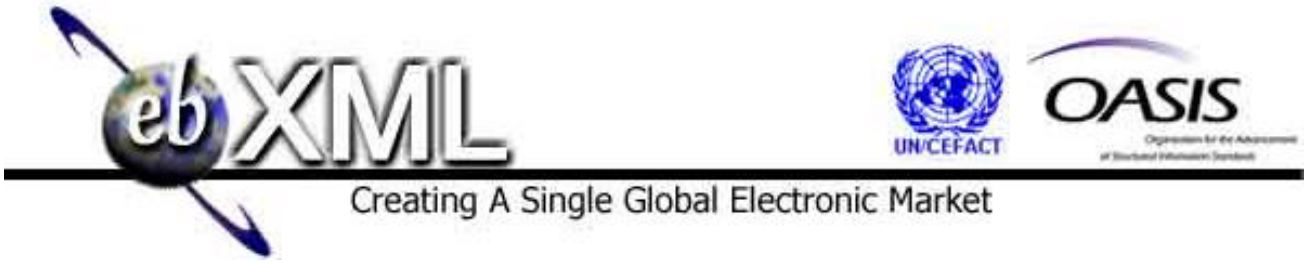

Document Assembly and Context Rules

ebXML Core Components

10 May 2001
Version 1.04

1 Status of this Document

This Technical Report document has been approved by the Core Component Project Team and has been accepted by the ebXML Plenary.

This document contains information to guide in the interpretation or implementation of ebXML concepts.

Distribution of this document is unlimited.

The document formatting is based on the Internet Society's Standard RFC format.

This version:

www.ebxml.org/specs/ebCCDOC.pdf

Latest version:

www.ebxml.org/specs/ebCCDOC.pdf

2 ebXML participants

We would like to recognize the following for their significant participation to the development of this document.

Editing team:	Mike Adcock, APACS
	Sue Probert, Commerce One
	James Whittle, e CentreUK
	Gait Boxman, TIE
	Thomas Becker, SAP
Team Leader:	
	Arofan Gregory, Commerce One
Vice Team Leader:	
	Eduardo Gutentag, SUN Microsystems
Contributors:	
	Eduardo Gutentag
	Arofan Gregory
	Matthew Gertner
	Martin Bryan
	Martin Roberts
	Lauren Wood
	Chris Nelson
	Todd Freter
	Mike Adcock

3 Table of Contents

1 Status of this Document 2
2 ebXML participants 3
3 Table of Contents 4
4 Introduction 5
4.1 Summary of Contents of Document 6
4.2 Related Documents 6
5 Document Assembly 7
6 Context Rules 8
7 XML-Based Rules Model 10
7.1 Rules Syntax 10
7.1.1 Notes on Assembly 14
7.1.2 Notes on Context 14
7.2 DTD for Assembly Documents 15
7.3 DTD for Context Rules Documents 16
8 Rule Ordering 19
9 Semantic Interoperability Document 20
9.1 DTD for Semantic Interoperability Document 20
10 Output Constraints 22
11 Appendix: Examples 23
11.1 Example of Assembly Rules document 23
11.2 Example of Context Rules Document 24
11.3 Example of Semantic Interoperability Document 26
12 References 27
13 Disclaimer 28
14 Contact Information 29
Copyright Statement 30

4 Introduction

The challenge of ebXML is to create a framework for automating trading partner interactions that is both:

- Sufficiently generic to permit implementation across the entire range of business processes (in various industries, geographical regions, legislative environments, etc.)
- Expressive enough to be more effective than ad hoc implementations between specific trading partners.

This specification document describes the way in which rules can be formed and/or derived, but is not a prescriptive specification. It is believed that rule mechanisms will be achieved in different ways within different implementations/solutions.

This document deals with two specific aspects of the task:

- The assembly of core component schemas into full business document schemas,
- The modelling of core components for business documents that provide useful building blocks for real-world trading scenarios and, at the same time, are open enough to take into account the wide variety of document formats required by organizations with differing business practices and requirements.

Complicating this situation is the need for interoperability: companies must be able to communicate business documents effectively with minimum human intervention, even though the formats used may have a significantly different syntax.

Central to achieving this goal is the notion of context. Context provides a framework for adapting generic core components to specific business needs, while keeping the transformation process transparent so that the processing engine can find a useful set of common information for use by different trading partners. An example of a contextual category that is useful for business is industry: different industries will have different requirements for the syntax of core components. By starting with a generic core component and using context to derive a context-specific core component, we ensure that, at the very least, the information in the generic component will be useful when interacting with a trading partner in a different context (i.e. industry, region, etc.). This should be contrasted with the alternative: context-specific business documents that are not built from generic core components and therefore provide no common basis for interaction outside of that context.

In order to assemble full business documents from core components, rules are drawn specifying what components are to be included in the document, and how.

In order to generate a context-specific core component, rules are associated with different values for each of the context categories. This document presents a proposed syntax for these context rules, and a methodology for applying them, in order to achieve maximum reuse of existing XML software development tools and libraries.

The keywords MUST, MUST NOT, REQUIRED, SHALL, SHALL NOT, SHOULD, SHOULD NOT, RECOMMENDED, MAY, and OPTIONAL, when they appear in this document, are to be interpreted as described in RFC 2119.

4.1 Summary of Contents of Document

This specification describes the mechanism for assembling documents from the library of Core Components. It describes the process of refining the components to contain exactly the information required by a specific business context and describes the output of this process such that it enables interoperability independent of any syntax binding. This approach also lends itself to an automated comparison with other, similar document definitions created in other syntaxes. The provided specifications are;

- A syntax for providing the assembly rules, with a DTD and sample;
- A syntax for refining the assembled structures, and indicating specific context drivers, also with DTD and sample;
- A format for capturing the critical information about the final result, provided as an XML DTD.

4.2 Related Documents

As mentioned above, other documents provide detailed definitions of some of the components and their inter-relationship. They include ebXML Specifications on the following topics:

- [ebCNTXT] Context and Re-Usability of Core Components Ver 1.04
- [ebCCNAM] Naming Convention for Core Components Ver 1.04
- [ccOVER] Catalogue of Core Components Ver 1.04

5 Document Assembly

Document assembly is the rules-based process whereby Core Components are extracted from the repository and used to create a schema model. That can then be used to create an XML schema which, when appropriate, and after the application of any relevant Context Rules, can be used to validate the contents of a business document.

For example, a Purchase order schema may consist of two parties (buyer, seller), and a sequence of items. Purchase orders are not Core Components; they must be assembled out of Core Components found in the repository.

6 Context Rules

When a business process is taking place, the context can be specified by a set of contextual categories and their associated values. For example, if an auto manufacturer is purchasing paint from a chemicals manufacturer, the context values might be as follows:

Contextual Category	Value
Process	Procurement
Product Classification	Paint
Region (buyer)	France
Region (seller)	U.S.
Industry (buyer)	Not required (generic)
Industry (seller)	Retail

Rules indicate which context values (or combination thereof) must be present in order for them to be applied, as well as the action to be undertaken if a match occurs. Actions include adding additional information to a functional unit, making this information optional, required or eliminating optional information. We might, for instance, specify that addresses associated with organizations in the U.S. region be required to include a state (which might otherwise be optional). Note that these contextual changes are made individually to the Core Components that make up a business document, and not to the business document itself.

Despite this underlying simplicity, complications arise in certain cases that make real-world implementation of context rules extremely tricky. Broadly speaking, these complications relate to scenarios where two rules both match the context, but have conflicting results, or where different results are reached depending on the order in which matching rules are applied. The following examples illustrate these two cases (and refer to the sample context given above; see also see the document ebXML TR - Catalogue of Context Drivers Ver 1.04):

- One rule could require that if the buyer is in the U.S. region, product description should not be included in invoice line items. Another specifies that if the seller is in France, the product description (in French) shall be included.
- One rule could require that if the buyer's industry is automotive, the product category should be added to the invoice line items. Another specifies that if a product category information entity exists and the seller's industry is chemicals, an attribute should be added to the product category to indicate the toxicity of the products in the category. If the toxicity requirement were applied first, the attribute would not be added (since the product category was not yet present). The outcome therefore depends on the order in which the rules are applied.

The problem with these types of situations is not so much that there is no way to resolve them. It is rather that there are many possible solutions with no clear way of deciding

which to choose, and all are sufficiently complex to place a significant burden on the implementer.

Additional complications result from the potentially hierarchical nature of context values. For example, the possible values for region belong in a hierarchical space (e.g. continent, country, region, city, etc.). The region specification can therefore be very general or very specific. Since rules can match a general value (e.g. apply if the organization is in North America) or a specific value (e.g. apply if the organization is in Omaha, Nebraska), there must be some way of determining which rules to apply (any combination including all of them) if several match. This is because, in some cases, a specific rule may complement the general rule, while in others it may override it.

7 XML-Based Rules Model

The custom XML syntax for assembly and context rules presented in this document is designed to ensure an appropriate level of abstraction for the rules, and to allow them to be applied both manually and/or by programs.

7.1 Rules Syntax

The syntax is presented here in tabular form, to avoid tying the definition of the schemas it describes to a given schema language syntax. This table should be sufficiently expressive to permit the derivation of a corresponding schema definition in various concrete schema syntaxes (DTD, XML Schema, SOX, XDR, etc.). This syntax describes two XML schemas describing two classes of XML documents whose roots are, respectively, `<Assembly>` and `<ContextRules>`. They are presented here in a single table because there is conceptual commonality.

A specific rules file is thus an XML document conforming to one of these schemas.

The following values are allowed for the occurrence field:

Name	Meaning
Required	Must occur exactly once
Optional	May occur once at most
+	Required and may occur multiply
*	Optional and may occur multiply
(m,n)	Occurs at least m and at most n times

Names separated by the vertical bar (|) represent a disjunction (i.e. one and only one of the list of names may occur). For example, Apple|Orange|Banana indicates that either an Apple or an Orange or a Banana may occur in this location.
Names prefixed with the commercial at sign (@) are represented as attributes in the XML instance (and the leading @ is removed from the attribute name).

Name	Type	Occurrence	Default	Description
Assembly				
Assemble	complex	+		List of assembled Core Components
@name	string	optional		Name of collection of assembled document schemas.
@version	string	optional		Version of the Assembly Rules document.
Assemble				
CreateElement	complex	+		List of Core Components

CreateGroup	complex	*		Create a group of elements
@name	string	required		Name of the document schema being assembled
CreateGroup				
@type	enum	default	sequence	Type of group to be created (the only permitted values are ‘sequence’ and ‘choice’)
CreateGroup	complex	*		Create a group of elements
CreateElement	complex	*		Create an Element
UseElement	complex	*		Use the named element from among the children of the element being created.
Annotation	complex	*		Insert Annotation
CreateElement				
Type	string	optional		Type of element to be created
MinOccurs	string	optional		Minimum occurrences for the element created
MaxOccurs	string	optional		Maximum occurrences for the element created. One possible value (other than integer) is ‘unbounded’.
@id	ID	required		Id of the created element
@idref	IDREF	optional		Reference to the ID of another created element
Name	string	required		Name of the element to be assembled
@location	UUID|URI	required		Location of the element to be assembled (i.e. query to the registry)
Rename	EMPTY	optional		Renames children of the created element
Annotation	complex	*		Insert Annotation
Rename				
@from	string	required		Original name of the child element being renamed
@to	string	required		New name of the child being renamed
ContextRules				
Rule	complex	+		List of rules to be applied
@version	string	optional		Version of the ContextRules document.

Rule				
@apply	enum	default	exact	(See below)
Condition	complex	required		When rule should be run
Action	complex	+		What happens when rule is run
@order	integer	default	0	Defines order for running rules. Rules with higher value for order are run first
Taxonomy	EMPTY	+		List of taxonomies used in a Rule that employs hierarchical conditions.
Taxonomy				
@ref	URI	Required		Pointer to a taxonomy.
Condition				
@Test	string	Required		Boolean expression testing whether the rule should be run. Uses XPath syntax [XPATH]
Action				
@applyTo	string	Required		Node to apply action to
Add|Subtract| Occurs	complex	+		List of modifications to content model
Add				
MinOccurs	integer	default	1	Minimum number of times that the new instance must occur
MaxOccurs	integer	default	1	Maximum number of times that the new instance can occur
@before	string	optional		Specifies before which child the addition should occur.
@after	string	optional		Specifies after which child the subtraction should occur.
Element	complex	optional		Adds a new element to the content model.
Attribute	complex	optional		Adds a new attribute to the content model
Annotation	complex	*		Insert Annotation
Subtract				
Element	complex	optional		Removes an element from the content model.
Attribute	Complex	optional		Removes an attribute from the content model
Occurs				

Element	complex	required		Changes an optional element to required.
MinOccurs	integer	optional	1	Overrides the minimum number of occurrences for this Element.
MaxOccurs	integer	optional	1	Overrides the maximum number of occurrences for this Element.
Element				
Nameame	string	required		Name of element to be modified
Type	string	optional		Type of element, required only if contained in an Add tag
Attribute	complex	*		Attribute(s) of this element
Annotation	complex	*		Insert Annotation
Attribute				
Name	string	optional		Name of attribute to be modified
Type	string	optional		Type of the attribute (e.g. ID, CDATA, enumerated list, etc.)
Use	required | optional | fixed | default	optional	required	Indicates whether required or optional, and if the latter whether fixed or defaulted
Value	string	optional		Indicates a fixed or defaulted value, or a value to be modified
UseElement				
Name	string	required		Name of the element being used
Annotation	complex	*		Insert Annotation
Comment				
	string	optional		Ubiquitous. Records comments about the rules document at the location it appears. It is not intended to be output in the result document.
Type				
	string	optional		Type in the output
MinOccurs				
	string	Optionl		Minimum number of occurrences in the output
MaxOccurs				

	string	Optional		Maximum number of occurrences in the output

7.1.1 Notes on Assembly

The MinOccurs and MaxOccurs elements in the CreateElement element specify the occurrence indicator that the created element will have in the resulting schema. Thus, an element created with `<MinOccurs>1</MinOccurs>` and `<MaxOccurs>1</MaxOccurs>` should be specified in the resulting schema as an element that must occur only once.

An `<Assembly>` may contain more than one assembled document schema. Whether a separate document is output for each assembled schema is implementation dependent.

7.1.2 Notes on Context

Several built-in variables are used to access context information. These variables correspond to the various context drivers identified in the document ebXML TR - Catalogue of Context Drivers Ver 1.04:

- Industry
- Business Process
- Product
- Geopolitical
- Official Constraints
- Role

All of these variables have values of type string.

The “Apply” attribute of the “Rule” element type is used for determining the behaviour of rules that use hierarchical value spaces. Possible values are “exact” (match only if the value in the provided context is precisely the same as that specified in the rule) and “hierarchical” (match if the value provided is the same or a child of that specified in the rule). For example, if the rule specifies the region “Europe”, the value “France” would match only if the “Apply” attribute is set to “hierarchical” (“exact” being the default).

The minOccurs and maxOccurs elements in Occurs are defaulted. If neither is present, the intent is to change an optional element into a required one (that is, it's a shortcut for `<MinOccurs>1</MinOccurs>, <MaxOccurs>1</MaxOccurs>`).

The <Attribute> element has four optional elements in its content model, of which at least one must be present. If the value of the applyTo attribute of Action is an attribute, there is no need to specify the Name again. If only Value is specified, the intention must be to add or subtract a given value from an attribute’s enumerated list.

Rules apply only to the source. For instance, given a source that contains an optional element type named ‘X’, a rule can be applied to rename ‘X’ to ‘Y’, but a rule to make ‘Y’ required, rather than ‘X’, would be illegal.

(also see [ebCNTXT] Context & Re-Usability of Core Components Ver1.04)

7.2 DTD for Assembly Documents

```
<!ELEMENT Assembly (Assemble+)>
<!ATTLIST Assembly
          version  CDATA #IMPLIED
          id       ID    #IMPLIED
          idref    IDREF #IMPLIED
>

<!ELEMENT Assemble (CreateElement|CreateGroup)+>
<!-- the name is the name of the schema that is created -->
<!ATTLIST Assemble
          name     CDATA #REQUIRED
          id       ID    #IMPLIED
          idref    IDREF #IMPLIED
>

<!ELEMENT CreateGroup
            (CreateGroup|CreateElement|UseElement|Annotation)+ >
<!ATTLIST CreateGroup
          type  (sequence|choice) "sequence"
          id       ID    #IMPLIED
          idref    IDREF #IMPLIED
>

<!ELEMENT CreateElement (Name?, Type?, MinOccurs?, MaxOccurs?,
(CreateGroup|Rename|UseElement|Condition|Annotation)*)>
<!-- you need either a Name sub-element and
an ID attribute, or just an IDREF attribute -->
<!-- max can be an integer or the word "unbounded" -->
<!ATTLIST CreateElement
          id       ID    #IMPLIED
          idref    IDREF #IMPLIED
          location CDATA #IMPLIED
>

<!ELEMENT Name    (#PCDATA)>
<!ELEMENT Type    (#PCDATA)>
<!ELEMENT MinOccurs (#PCDATA)>
<!ELEMENT MaxOccurs (#PCDATA)>

<!ELEMENT Rename  EMPTY>
<!ATTLIST Rename
          from     CDATA #REQUIRED
          to       CDATA #REQUIRED
          id       ID    #IMPLIED
          idref    IDREF #IMPLIED
>

<!ELEMENT UseElement  (Annotation|CreateGroup|UseElement)*>
<!ATTLIST UseElement
          name     CDATA #REQUIRED
          id       ID    #IMPLIED
          idref    IDREF #IMPLIED
>

<!ELEMENT Condition (Rename|CreateGroup|UseElement|CreateElement)+>
```

```
<!ATTLIST Condition
          test     CDATA #REQUIRED
          id       ID    #IMPLIED
          idref    IDREF #IMPLIED
>
<!ELEMENT Annotation  (Documentation | AppInfo)*>
<!ATTLIST Annotation
          id       ID    #IMPLIED
          idref    IDREF #IMPLIED
>

<!ELEMENT Documentation  (#PCDATA)>
<!ATTLIST Documentation
          id       ID    #IMPLIED
          idref    IDREF #IMPLIED
>

<!ELEMENT AppInfo  (#PCDATA)>
<!ATTLIST AppInfo
          id       ID    #IMPLIED
          idref    IDREF #IMPLIED
>
```

7.3 DTD for Context Rules Documents

```
<!ELEMENT ContextRules (Rule+)>
<!ATTLIST ContextRules
          version  CDATA #IMPLIED
          id       ID    #IMPLIED
          idref    IDREF #IMPLIED
>

<!ELEMENT Rule    (Taxonomy+, Condition+)>
<!ATTLIST Rule
          apply    (exact|hierarchical) “exact”
          order    CDATA #IMPLIED
          id       ID    #IMPLIED
          idref    IDREF #IMPLIED
>

<!ELEMENT Taxonomy    EMPTY>
<!-- this ref should be a URI -->
<!ATTLIST Taxonomy
          context  CDATA #REQUIRED
          ref      CDATA #REQUIRED
          id       ID    #IMPLIED
          idref    IDREF #IMPLIED
>


<!ELEMENT Condition  (Action|Condition|Occurs)+>
<!ATTLIST Condition
          test     CDATA #REQUIRED
          id       ID    #IMPLIED
          idref    IDREF #IMPLIED
```

```
>

<!ELEMENT Action  (Add|Occurs|Subtract|Condition|Comment|Rename)+>
<!ATTLIST Action
          applyTo  CDATA #REQUIRED
          id       ID    #IMPLIED
          idref    IDREF #IMPLIED
>

<!ELEMENT Add      ((MinOccurs?,MaxOccurs?,(Element?
                            |Attribute?))|CreateGroup|Annotation)+>
<!-- before and after refer either to the ID of the other element or
to its Xpath -->
<!ATTLIST Add
          before   CDATA #IMPLIED
          after    CDATA #IMPLIED
          id       ID    #IMPLIED
          idref    IDREF #IMPLIED
>

<!ELEMENT Rename  EMPTY>
<!ATTLIST Rename
          from     CDATA #REQUIRED
          to       CDATA #REQUIRED
          id       ID    #IMPLIED
          idref    IDREF #IMPLIED
>

<!ELEMENT CreateGroup  (Element)+>
<!ATTLIST CreateGroup
          type  (choice|sequence) “sequence”
          id       ID    #IMPLIED
          idref    IDREF #IMPLIED
>

<!ELEMENT Element      (Name, Type?, (Attribute)*, (Annotation)*)>
<!ATTLIST Element
          id       ID    #IMPLIED
          idref    IDREF #IMPLIED
>

<!ELEMENT Attribute      (Name?, Type?, Use?,
                                  Value?, (Annotation)*)>
<!ATTLIST Attribute
          id       ID    #IMPLIED
          idref    IDREF #IMPLIED
>

<!ELEMENT Use (#PCDATA)>
<!ELEMENT Value (#PCDATA)>

<!ELEMENT Annotation  (Documentation | AppInfo)*>
<!ATTLIST Annotation
          id       ID    #IMPLIED
          idref    IDREF #IMPLIED
>
```

```
441
442   <!ELEMENT Documentation  (#PCDATA)>
443   <!ATTLIST Documentation
444             id       ID    #IMPLIED
445             idref    IDREF #IMPLIED
446   >
447   <!ELEMENT AppInfo  (#PCDATA)>
448   <!ATTLIST AppInfo
449             id       ID    #IMPLIED
450             idref    IDREF #IMPLIED
451   >
452
453   <!ELEMENT Occurs (MinOccurs?, MaxOccurs?,(Element+))>
454   <!ATTLIST Occurs
455             id       ID    #IMPLIED
456             idref    IDREF #IMPLIED
457   >
458
459   <!ELEMENT Subtract (Element | Attribute)+>
460   <!ATTLIST Subtract
461             id       ID    #IMPLIED
462             idref    IDREF #IMPLIED
463   >
464
465   <!ELEMENT Name    (#PCDATA)>
466   <!ELEMENT Type    (#PCDATA)>
```

8 Rule Ordering

There are two mechanisms for determining the order in which context rules should be applied. The first is document order, that is, the order in which the rules appear in the Rules document. The second is an explicit “Order” attribute that can be used to force a given order on a set of rules. It's an error for two rules have the same order. Users should be careful not to issue rules in an order that would preclude their execution (for instance, adding an attribute to an element that has not been added yet by the rules). Applications must issue error messages when such a situation is encountered, rather than silently ignoring it.

9 Semantic Interoperability Document

This section specifies an XML document format, the Semantic Interoperability Document that a processor applying assembly rules and context rules within a single context can output. This serves two purposes:

- It creates a syntax-neutral output format, so that two processors working with different syntax mappings could determine the semantic equivalence of their context rules by comparing the output when expressed in this form.
- It provides a mechanism for mapping from a syntax-specific output back to the syntax-neutral one, using techniques such as UUID pointers or Xpath expressions, enabling implementation using existing tools.

9.1 DTD for Semantic Interoperability Document

The semantic interoperability document type is expressed in the following DTD:

```
<!-- Semantic Interoperability Document Definition -->
<!-- the Document element holds metadata about the document: -->

<!ELEMENT Document (Taxonomy+, Assembly, ContextRules?,Component+) >

<!-- - Taxonomy points to the specific context that, combined with
context rules and assembly rules, produced the specific instance.
The content of the Taxonomy element is the value or values specified
from the referenced context taxonomy.
- Assembly references the assembly that produced the instance.
- ContextRules references the context rules that produced the
instance.-->
<!ATTLIST Document
          name     CDATA #IMPLIED
          UUID     CDATA #IMPLIED
>

<!ELEMENT Taxonomy (#PCDATA)>
<!ATTLIST Taxonomy
          context  CDATA #REQUIRED
          ref      CDATA #REQUIRED
          UUID     CDATA #IMPLIED
>

<!ELEMENT Assembly EMPTY>
<!--     For each specified contextual value for the document, you must
supply a context name and a value, expressed as the name of the context
driver (the top level of the context hierarchy), an equals sign, and
one or more values enclosed in single quotes. For example:

        value="Industry='Aerospace' Geopolitical='United States'"

Note that ranges in the value position are indicated by hyphens and
that path expressions are valid values. Lists of values may be
indicated by using commas or pipes, with or without whitespace.
-->
<!ATTLIST Assembly
```

```
527             name     CDATA #REQUIRED
528             value    CDATA #REQUIRED
529             UUID     CDATA #IMPLIED
530   >
531   <!ELEMENT ContextRules EMPTY>
532   <!ATTLIST ContextRules
533             name     CDATA #REQUIRED
534             value    CDATA #REQUIRED
535             UUID     CDATA #IMPLIED>
536
537   <!ELEMENT Component (Component | Group)*>
538
539   <!--    - Type attribute must be included if the element is of a simple
540   type. If it is not provided, the name value is assumed to be the same
541   as the complex type name.
542           - Occurrence applies to the component itself and indicates how
543   often it occurs in the final schema.It must be one of the following:
544                   [no value is "one and only one"]
545                   ?
546                   +
547                   *
548                   n,m where n is minimum and m is maximum
549
550
551   - Sequence applies to the children of the component. It is information
552   in the context rules that must be kept, even if not all syntaxes need
553   it or support it. Values should be:
554                   FollowedBy: the order in which the children are
555   specified is important, and is
556   the order in which they will be specified in the final schema.
557                   AnyOrder: the order in which the children are specified
558   is not important, since the final schema will allow them in any order.
559   All of the children must be present in a document written according to
560   the final schema.
561                   Choice: the order in which the children are specified
562   is not important. Only one of the children is allowed in a document
563   written according to the final schema.
564    -->
565   <!ATTLIST Component
566             name       CDATA #REQUIRED
567             type       CDATA #IMPLIED
568             occurrence CDATA #IMPLIED
569             sequence   CDATA #IMPLIED
570             UUID       CDATA #IMPLIED
571   >
572
573   <!-- The Group element functions as a way of describing the structural
574   relationships among nested, unnamed groups of child components. The use
575   of its attributes are the same as for the Component elements.
576   -->
577   <!ELEMENT Group (Component | Group)*>
578   <!ATTLIST Group
579             occurrence CDATA #IMPLIED
580             sequence   CDATA #IMPLIED
581   >
```

10 Output Constraints

Documents produced through the application of Assembly and Context Rules must contain information regarding which rules and context were used as metadata.

11 Appendix: Examples

11.1 Example of Assembly Rules document

```
<?xml version="1.0"?>
<!DOCTYPE Assembly SYSTEM "assembly.dtd">
<Assembly version="1.0">
  <Assemble name="PurchaseOrder" id=”PO”>
    <CreateGroup>
      <CreateElement location="UUID" id="Buyer">
        <Name>Buyer</Name>
        <Type>PartyType</Type>
        <CreateGroup>
          <UseElement name="Name">
          </UseElement>
          <UseElement name="Address">
            <CreateGroup id="fred">
              <CreateGroup type="choice">
                <UseElement name="BuildingName">
                </UseElement>
                <UseElement name="BuildingNumber">
                </UseElement>
              </CreateGroup>
              <UseElement name="StreetName">
              </UseElement>
              <UseElement name="City">
              </UseElement>
              <UseElement name="State">
              </UseElement>
              <UseElement name="ZIP">
              </UseElement>
              <UseElement name="Country">
              </UseElement>
            </CreateGroup>
          </UseElement>
        </CreateGroup>
        <Condition test="$Geopolitical='United States'">
          <Rename from="address" to="addressUS"/>
          <Rename from="Place" to="City"/>
          <Rename from="address/County" to="State"/>
          <Rename from="address/PostalCode" to="ZIP"/>
        </Condition>
      </CreateElement>
      <CreateElement id="Seller" location="UUID">
        <Name>Seller</Name>
        <Type>PartyType</Type>
      </CreateElement>
    </CreateGroup>
    <CreateElement location="UUID" id="Item">
      <Name>Item</Name>
      <Type>ItemType</Type>
      <MinOccurs>1</MinOccurs>
      <MaxOccurs>unbounded</MaxOccurs>
    </CreateElement>
  </Assemble>
```

```
  <Assemble name="PurchaseOrderReceipt" id=”POR”>
    <CreateGroup>
      <CreateElement idref="Seller">
      </CreateElement>
      <CreateElement idref="Buyer">
      </CreateElement>
    </CreateGroup>
    <CreateElement idref="Item">
    </CreateElement>
    <CreateElement location="UUID" id="Ack">
      <Name>Acknowledgment</Name>
      <Type>AckType></Type>
    </CreateElement>
  </Assemble>
</Assembly>
```

11.2 Example of Context Rules Document

```
<?xml version="1.0"?>
<!DOCTYPE ContextRules SYSTEM "contextrules.dtd">
<ContextRules id=”CalAer”>
  <Rule apply="hierarchical">
    <Taxonomy context="Geopolitical"
      ref="http://ebxml.org/classification/ISO3166"/>
    <Taxonomy context="Industry"
      ref="http://ebxml.org/classification/industry/aviation"/>
    <Condition test="$Geopolitical='United States'">
      <Action applyTo="//Buyer/Address">
        <Occurs>
          <Element >
            <Name>State</Name>
          </Element>
        </Occurs>
        <Add after="@id='fred'">
          <CreateGroup type="choice">
            <Element >
              <Name>Floor</Name>
              <Type>string</Type>
            </Element>
            <Element >
              <Name>Suite</Name>
              <Type>string</Type>
            </Element>
          </CreateGroup>
        </Add>
        <Condition
          test="$Geopolitical='California' and$Industry='Aerospace'">
          <Occurs>
            <Element >
              <Name>ZIP</Name>
            </Element>
          </Occurs>
        </Condition>
      </Action>
    </Condition>
  </Rule>
```

```
<Rule order="10">
  <Taxonomy context="Geopolitical"
  ref="http://ebxml.org/classification/ISO3166"/>
  <Condition test="$Business Process='RFQ'">
    <Condition test="Industry='Insurance'">
      <Action applyTo="//Party">
        <Add before="Address">
          <Element >
            <Name>QualifyingInfo</Name>
            <Type>QualifyingInfo</Type>
            <Attribute>
              <Name>Privacy</Name>
              <Type>yes | no</Type>
              <Use>default</Use>
              <Value>yes</Value>
            </Attribute>
            <Attribute>
              <Name>Accuracy</Name>
              <Type>CDATA</Type>
              <Use>required</Use>
            </Attribute>
            <Annotation>
              <Documentation>What this element is for.
              </Documentation>
            </Annotation>
          </Element>
        </Add>
      </Action>
    </Condition>
    <Condition test="$Industry='Travel'">
      <Action applyTo="//Party">
        <Subtract>
          <Attribute >
            <Name>TaxIdentifier</Name>
          </Attribute>
        </Subtract>
      </Action>
    </Condition>
  </Condition>
</Rule>
<Rule>
  <Taxonomy context="Industry"
    ref="http://ebxml.org/classification/Industry/Automotive"/>
  <Condition test="$Industry='Automotive'">
    <Action applyTo="//QualifyingInfo">
      <Add>
        <Element >
          <Name>DrivingRecord</Name>
          <Type>DrivingRecord</Type>
        </Element>
        <Element >
          <Name>CarDescription</Name>
          <Type>CarDescription</Type>
        </Element>
        <Element >
          <Name>DrivingHabits</Name>
```

```
            <Type>DrivingHabits</Type>
          </Element>
        </Add>
        <Rename from="@Convictions" to="@DrivingConvictions"/>
      </Action>
      <Action applyTo=”//QualifyingInfo/@Convictions”>
        <Add>
          <Attribute>
            <Value>Unknown</Value>
          </Attribute>
        </Add>
      </Action>
    </Condition>
  </Rule>
</ContextRules>
```

11.3 Example of Semantic Interoperability Document

This example assumes a US address, and the California/Aerospace example from above.

```
<?xml version=”1.0”?>
<!DOCTYPE Document SYSTEM “sid.dtd”>
<Document>
  <Taxonomy context="Geopolitical"
    ref="http://ebxml.org/classification/ISO3166">Region
  </Taxonomy>
  <Assembly name="PurchaseOrder" value=”Geopolitical=’United States’”/>
  <ContextRules name=”CalAer” value="Industry='Aerospace'
    Geopolitical='United States'"/>
  <Component name=”PurchaseOrder” sequence=”FollowedBy”>
    <Component name=”Buyer” sequence=”FollowedBy”>
      <Component name=”Address” sequence=”FollowedBy”>
      <Group sequence=”Choice”>
        <Component name=”BuildingName”/>
          <Component name=”BuildingNumber”/>
        </Group>
        <Group sequence=”Choice”>
          <Component name=”Floor”/>
          <Component name=”Suite”/>
        </Group>
        <Component name=”City”/>
        <Component name=”State”/>
        <Component name=”ZIP”/>
        <Component name=”Country”/>
      </Component>
    </Component>
    <Component name=”Seller”/>
    <Component name=”Item” occurrence=”+”/>
  </Component>
</Document>
```

12 References

[XPATH] http://www.w3.org/TR/xpath

13 Disclaimer

The views and specification expressed in this document are those of the authors and are not necessarily those of their employers. The authors and their employers specifically disclaim responsibility for any problems arising from correct or incorrect implementation or use of this design.

14 Contact Information

Team Leader

Name	Arofan Gregory
Company	Commerce One
Street	Vallco Parkway
City, state, zip/other	Cupertino, CA
Nation	US
Phone:	
Email:	arofan.gregory@commerceone.com

Vice Team Lead

Name	Eduardo Gutentag
Company	SUN Microsystems
Street	17 Network Circle – UMPK17-102
City, state, zip/other	Menlo Park, California
Nation	US
Phone:	+1-650-786-5498
Email:	Eduardo.Gutentag@eng.sun.com

Editor

Name	Gait Boxman
Company	TIE
Street	Beech Avenue 161
City, state, zip/other	Amsterdam (Schiphol-Rijk)
Nation	The Netherlands
Phone:	
Email:	gait.boxman@tie.nl

Copyright Statement

Copyright © UN/CEFACT and OASIS, 2001. All Rights Reserved.