ebXML specification for the application of XML based assembly and context rules

ebXML Core Components

Version: 1.0

Status: FOR APPROVAL

Date: 15 February 2001

1 Status of this Document

This document specifies an ebXML draft specification for the eBusiness community.

Distribution of this document is unlimited.

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

This version can be found at:

http://www.ebxml.org/working/project_teams/core_components/latest.htm

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:
Martin Bryan
Lauren Wood
Tom Warner
Jim Dick
Rob Jeavons
David Connelly
Mike Adcock
Eduardo Gutentag
Matthew Gertner
Todd Freter
Henrik Reiche
Chris Nelson
Martin Roberts
Samantha Rolefes
Stig Korsgaard

3 Table of Contents

4 Introduction

The ebXML standards effort is faced with a daunting challenge: to create a framework for automating trading partner interactions that is at once sufficiently generic to permit implementation across the entire range of business processes (in various industries, geographical regions, legislative environments, etc.), while remaining expressive enough to be more effective than ad hoc implementations between specific trading partners. This document deals with two specific aspects of this task:

1. The assembly of Core Component schemas into full business document schemas, and
2. The modeling 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 geographic region: different industries will have different requirements for the syntax and semantics 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 these context rules, and a methodology for applying them, in order to achieve maximum reuse of existing XML software development tools and libraries.

4.1 Summary of Contents of Document

This specification describes the mechanism for assembling documents from the library of Core Components, refining the components to contain exactly the information required by a specific business context and describes the output of this process in a fashion that 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.

What is *not* provided in this document is an exact specification of mappings to any particular syntax, this not being a critical ingredient for achieving interoperability in the fashion provided for.

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;

- ebXML The role of context in the re-usability of Core Components and Business Processes Ver 1.0
- ebXML Naming conventions for Core Components and Business Processes Ver1.0
- ebXML Initial catalogue of Core Components Ver1.0

5 Document Assembly

Document assembly is the rules-based process whereby various Core Component schemas are extracted from the repository and put together in a schema for a full business document, in a specified order, with the appropriate occurrence indicators and with the right choice of element content (sequence, choice, etc.)

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 and Context Rules

The basic concept of context rules is actually quite simple. 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 an additional field to a functional unit, making an optional field required or eliminating an optional field. 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):

- One rule could rquire 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

In order to ensure the appropriate level of abstraction for the rules, and to allow them to be applied both manually and/or by programs, we propose to design a custom XML syntax for assembly and context rules

7.1 Proposed Rules Syntax

The proposed syntax takes the form of 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. In order to avoid tying the definition of these schemas to a given schema language syntax, they are presented in tabular form. 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.).

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.
Context	complex	required		List of contexts used in this assembly
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.
CreateElement				
@Type	enum	optional		Type of element to be created
@minOccurs	integer	optional		Minimum occurrences for the element created
@maxOccurs	integer	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	GUID|URI	required		Location of the element to be assembled (i.e. query to the registry)
Rename	EMPTY	optional		renames children of the created element
ApplySequence	complex	+		Creates a sequence of elements in the result document.
ApplyChoice	complex	+		Creates a choice of elements in the result document.
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.
context	complex	required		List of contexts used in this 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 the same XPath syntax as XSLT [XSLT]
Action				
@ApplyTo	string	Required		Node to apply action to
Add|Subtract|Occur	complex	+		List of modifications to content model
Add				
@MinOccurs	integer	default	1	Minimum number of times that the new field must occur
@MaxOccurs	integer	default	1	Maximum number of times that the new field can occur
@Before	string	optional		Specifies before which child the addition should occur.
@After	string	optional		Specifies after which child the substraction should occur.
Field	complex	required		Adds a new field to the content

				model.
NewElement	complex	required		Adds a new element to the content model.
Subtract				
Field	complex	required		Removes a field from the content model .
Occur				
Field	complex	required		Changes an optional field to required.
@minOccurs	integer	default	1	Overrides the minimum number of occurrences for this Field
@maxOccurs	integer	default	1	Overrides the maximum number of occurrences for this Field
Field				
@Name	string	required		Name of field to be modified
@Type	string	optional		Type of field, required only if contained in an Add tag
UseElement				
Name	string	required		Name of the element being used
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.
Context				
Region	string	*		Value of region context used in this rules document.
Industry				Value of industry context used in this rules document.
Process				Value of process context used in this rules document.
Product				Value of product context used in this rules document.
legislative				Value of legislative context

				used in this rules document.
role				Value of role context used in this rules document.

7.1.1 Notes on Assembly

The @minOccurs and @maxOccurs attributes on the Create element specify the occurrence indicator that the created element will have in the resulting schema. Thus, an element created with @min='1' @max='1' 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.

Issue: Do we need to create attributes? If so, how?

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 by the CCWG:

- Industry
- Process
- Product
- Region
- Legislative
- Role

All of these variables have values of type string.

The "Apply" attribute of the "Rule" element type is used for determining the behavior 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 attributes of Field are defaulted. If neither is present, the intent is to change an optional field into a required one (that is, it's a shortcut for minOccurs="1", maxOccurs="1").

7.2 DTD for Assembly Documents

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

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

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

<!ELEMENT CreateElement (Name?,
(CreateGroup|Rename|UseElement|Condition|Annotation)*)>
<!ATTLIST CreateElement
          type     CDATA  #IMPLIED
          minOccurs      NUMBER #IMPLIED
          maxOccurs      CDATA  #IMPLIED
          id       ID     #IMPLIED
          idref    IDREF  #IMPLIED
          location CDATA  #IMPLIED
>
<!-- 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" -->

<!ELEMENT Name    (#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
>
```

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   NUMBER   #IMPLIED
          id      ID       #IMPLIED
          idref   IDREF    #IMPLIED
>

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

<!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      (Field|CreateGroup|Annotation)+>
<!ATTLIST Add
          before  CDATA  #IMPLIED
          after   CDATA  #IMPLIED
          id      ID    #IMPLIED
          idref   IDREF #IMPLIED

>
<!-- before and after refer to the ID of the other element -->

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

<!ELEMENT CreateGroup  (Field)+>
<!ATTLIST CreateGroup
          type  (choice|sequence) sequence
          id      ID    #IMPLIED
          idref   IDREF #IMPLIED
>
```

```
<!ELEMENT Field      (Annotation)*>
<!ATTLIST Field
          name    CDATA   #REQUIRED
          type    CDATA   #IMPLIED
          id      ID    #IMPLIED
          idref   IDREF #IMPLIED
>
<!-- why isn't name an IDREF that points to the ID of the element? -->

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

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

<!ELEMENT Occurs (Field+)>
<!ATTLIST Occurs
          minOccurs  NUMBER #IMPLIED
          maxOccurs  CDATA  #IMPLIED
          id      ID    #IMPLIED
          idref   IDREF #IMPLIED
>

<!ELEMENT Subtract (Field+)>
<!ATTLIST Subtract
          id      ID    #IMPLIED
          idref   IDREF #IMPLIED
>
```

7.4 Example of Assembly Rules document

```
<?xml version="1.0"?>
<!DOCTYPE Assembly SYSTEM "assembly.dtd">
<Assembly version="1.0">
  <Assemble name="PurchaseOrder">
    <CreateGroup>
      <CreateElement type="PartyType" location="GUID" id="Buyer">
        <Name>Buyer</Name>
        <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="Region='UK'">
          <Rename from="address" to="addressUK"/>
          <Rename from="City" to="Place"/>
          <Rename from="address/State" to="County"/>
          <Rename from="address/ZIP" to="PostalCode"/>
        </Condition>
      </CreateElement>
      <CreateElement type="PartyType" id="Seller" location="GUID">
        <Name>Seller</Name>
      </CreateElement>
    </CreateGroup>
    <CreateElement minOccurs="1" maxOccurs="unbounded"
      type="ItemType" location="GUID" id="Item">
      <Name>Item</Name>
    </CreateElement>
  </Assemble>
  <Assemble name="PurchaseOrderReceipt">
    <CreateGroup>
      <CreateElement idref="Seller">
      </CreateElement>
      <CreateElement idref="Buyer">
      </CreateElement>
    </CreateGroup>
    <CreateElement idref="Item">
    </CreateElement>
    <CreateElement type="AckType" location="GUID"
      id="Ack">
      <Name>Acknowledgment</Name>
    </CreateElement>
  </Assemble>
</Assembly>
```

7.5 Example of Context Rules Document

```
<?xml version="1.0"?>
<!DOCTYPE ContextRules SYSTEM "contextrules.dtd">
<ContextRules>
  <Rule apply="hierarchical">
    <Taxonomy context="Region"
      ref="http://ebxml.org/classification/ISO3166"/>
    <Taxonomy context="Industry"
      ref="http://ebxml.org/classification/industry/aviation"/>
```

```
475       <Condition test="Region='United States'">
476         <Action applyTo="Buyer/Address">
477           <Occurs>
478             <Field name="State">
479             </Field>
480           </Occurs>
481           <Add after="@id='fred'">
482             <CreateGroup type="choice">
483               <Field name="Floor" type="string">
484               </Field>
485               <Field name="Suite" type="string">
486               </Field>
487             </CreateGroup>
488           </Add>
489           <Condition test="Region='California' and Industry='Aerospace'">
490             <Occurs>
491               <Field name="ZIP">
492               </Field>
493             </Occurs>
494           </Condition>
495         </Action>
496       </Condition>
497     </Rule>
498     <Rule order="10"><Taxonomy context="Region"
499       ref="http://ebxml.org/classification/ISO3166"/>
500       <Condition test="Process='RFQ'">
501         <Condition test="Industry='Insurance'">
502           <Action applyTo="Party">
503             <Add before="Address">
504               <Field name="QualifyingInfo" type="QualifyingInfo">
505                 <Annotation>
506                   <Documentation>What this element is for.
507                   </Documentation>
508                 </Annotation>
509               </Field>
510             </Add>
511           </Action>
512         </Condition>
513         <Condition test="Industry='Travel'">
514           <Action applyTo="Party">
515             <Subtract>
516               <Field name="@TaxIdentifier">
517               </Field>
518             </Subtract>
519           </Action>
520         </Condition>
521       </Condition>
522     </Rule>
523     <Rule>
524       <Taxonomy context="Industry"
525         ref="http://ebxml.org/classification/Industry/Automotive"/>
526       <Condition test="Industry='Automotive'">
527         <Action applyTo="QualifyingInfo">
528           <Add>
529             <Field name="DrivingRecord" type="DrivingRecord">
```

```
            </Field>
            <Field name="CarDescription" type="CarDescription">
            </Field>
            <Field name="DrivingHabits" type="DrivingHabits">
            </Field>
          </Add>
          <Rename from="@Convictions" to="@DrivingConvictions"/>
        </Action>
      </Condition>
    </Rule>
</ContextRules>
```

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

As a useful possible output of a processor taking a single context and applying assembly rules and context rules, we have specified an XML document format, the Semantic Interoperability Document. 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.

This document type is expressed in the following DTD:

```
<!-- Semantic Interoperability Document Defintion -->
<!element Document (Taxonomy+, Assembly, ContextRules?, Component+) >
<!attlist Document
                Name CDATA #REQUIRED
                GUID CDATA #IMPLIED>
<!-- the Document element holds metadata about the document:

- 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.

-->

<!element Taxonomy (#PCDATA)>
<!attlist Taxonomy
                context CDATA #REQUIRED
                ref CDATA #REQUIRED
                UUID CDATA #IMPLIED>
<!element Assembly EMPTY>
<!attlist Assembly
                Name CDATA #REQUIRED
                Value CDATA #REQUIRED
                UUID CDATA #IMPLIED>
<!element ContextRules EMPTY>
<!attlist ContextRules
                Name CDATA #REQUIRED
                Value CDATA #REQUIRED
                UUID CDATA #IMPLIED>
<!element Component (Component | Group)*>
<!attlist Component
                Name CDATA #REQUIRED
                Type CDATA #IMPLIED
                Occurrence CDATA #REQUIRED
                Sequence CDATA #REQUIRED
                UUID CDATA #IMPLIED>
```

```
<!--    - Type attribute must be included if the element is of a simple
type. If it is not provided, the name
value is assumed to be the same as the complex type name.
        - Occurrence applies to the component itself and indicates how
often it occurs in the final schema.
It must be one of the following:
                [no value is "one and only one"]
                ?
                +
                *
                n,m where n is minimum and m is maximum


- Sequence applies to the children of the component. It is information
in the context rules that must be kept, even
if not all syntaxes need it or support it. Values should be:
                FollowedBy: the order in which the children are
specified is important, and is
the order in which they will be specified in the final schema.
                AnyOrder: the order in which the children are specified
is not important, since the
final schema will allow them in any order. All of the children must be
present in a document written
according to the final schema.
                Choice: the order in which the children are specified
is not important. Only one of the
children is allowed in a document written according to the final
schema.
 -->
<!element Group (Component | Group)*>
<!attlist Group
                Occurrence CDATA #REQUIRED
                Sequence CDATA #REQUIRED
                >
<!-- The Group element functions as a way of describing the structural
relationships among nested, unnamed groups of child components. The use
of its attributes are the same as for the Component elements.
-->
```

10 Output Constraints

Documents produced through the application of Assembly and Context Rules must carry on information regarding what rules were used and what context as metadata. An example of this metadata can be seen in the Interoperability Document, in the Taxonomy, Assembly and ContextRules elements therein.

11 References

[XSL] http://www.w3.org/Style/XSL

12 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.

13 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	
city, state, zip/other	California
Nation	US
Phone:	
EMail:	

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

14 Copyright Statement