OASIS ebXML Registry-Repository TC

CC-Review SubCommittee

Report for Serialization and Storage of Core Components (CC's) and Business Information Entities (BIE's) within ebXML Registry Repository facilities.

Version 0.90 (Draft until version 1.0)
Jan 12, 2005

Editors:
Duane Nickull – dnickull@adobe.com

Contributors:

Farrukh Najmi
Monica Martin
Diane Lewis
Matt Mackenzie

STATUS OF THIS DOCUMENT

This document is not complete. It captures aspects and issues associated with how to use UN/CEFACT Core Components and Business Information Entities within an ebXML Registry-Repository.

This document is non normative.

Table of Contents

1.0 Abstract

This best practices paper addressed the set of requirements for the serialization and storage formats for UN/CEFACT Core Components (CC's) and Business Information Entities (BIE's), both basic and aggregate, within ebXML Registry-Repository facility(s) as expressed in UN/CEFACT's Core Component Technical Specification v 2.01. The results are expressed as a set of recommendations intended for audiences addressing this problem.

This paper is a "Best Practice" and not a normative specification.

Readers are encouraged to read the requirements document since the information is not duplicated herein.

2.0 Statement of Work

1.1 Background

In May of 2001, the United Nations CEFACT (Center for Facilitation of Trade and Commerce) and OASIS (Organization for the Advancement of Structured Information Systems) delivered a set of specifications called ebXML (Electronic Business XML).

Version 1.0 of ebXML specified a methodology for creating and managing a set of reusable data element metadata called Core Components. The Core Components work continued under the auspices of UN/CEFACT and is guided by the United Nations Unified Modelling Methodology (UMM), a methodology that uses the Unified Modelling Language (UML) as it's syntax. The Core Components work has advanced to a 2.0 version of the specification. Subsequent revisions are expected however the work hereinafter accounts for such and will not become deprecated in the event the CCTS advances.

The Core Components Technical Specification addresses many aspects of Data Element Metadata however it does not address a format for serialization or storage within an ebXML Registry. As per the ebXML Technical Architecture, there is a normative requirement for Core Components to be stored, referenced and retrieved from within ebXML Registry-Repositories.

A requirement has been identified whereby users of the core components would also use the Core Components and associated Business Information Entities outside the environment of a metadata storage facility. This requirement mandates the need for a standardized serialization of the core components and business information entities. While it had been originally envisioned that the ebXML Registry Information Model (RIM) would sufficiently be suited for storage of the core components and business information entities, it cannot always be presumed that an ebXML registry will be present or call-able, therefore an independent serialization has been requested by many potential implementer's.

The ebXML Registry-Repository Technical Committee, operating under the auspices of OASIS, formed a sub-technical Committee to work on an in-term "*Best Practices*" solution until such time as the UN/CEFACT Core Components work may advance to include a normative specifications for storage and retrieval. It is hoped that this document may also provide input for that process.

2.2 Approach and Solution

The proposed methodology to solve the problem is based on a four point plan of action.

1. Document the requirements from stakeholders of the data elements. Ensure all stakeholders were represented and their requirements well documented as per the UMM and Business Collaboration Framework (BCF) methodologies. This step was accomplished as of April 27, 2004 and agreed at the face to face meeting of the OASIS ebXML Registry TC in New Orleans, LA April 27, 2004. The document is available at http://www.oasis-open.org/apps/org/workgroup/regrep/regrep-cc-review/download.php/6349/OASIS_ebXML_Registry-CC_Review_Requirements-v0.92.doc

2. Review and reconcile the UN/CEFACT Core Components and ebXML Registry Information Model (RIM) and Registry Specification Schema (RSS) data models and derive a syntax neutral data element metadata model. Take careful steps to ensure the model will meet all the current and future functional requirements of the stakeholders.

3. Develop a serialization (expression) of the Core Components Metadata model in XML. Account for future forwards and backwards compatibility and ease of implementation from a programmers' perspective and ease of use from a users perspective.

4. Develop a model and methodology for storing the Core Component Metadata within instances of ebXML Registry-Repositories.

2.3 Requirements for Core Component Serialization and Storage

The full requirements work of this Technical Sub-committee is available at http://www.oasis-open.org/apps/org/workgroup/regrep/regrep-cc-review/download.php/6349/OASIS_ebXML_Registry-CC_Review_Requirements-v0.92.doc. Below is a short excerpt.

In order for data elements to be placed and managed within an ebXML registry, they must be serialized into a format that allows them to be bound to the Registry. Additionally, they must be serialize-able into a format that facilitates entry and retrieval to and from a Registry system. A serialization is a format, which includes both the syntax and the taxonomy for expressing a Data Element. There are no formal standards for defining a format for such a binding or serialization. The UN/CEFACT Core Components Working Group defined a core component and BIE data model that was used as the basis for this work.

> **Definition:** Data Element Metadata (DEM) is a generic descriptor that covers both UN/CEFACT Core Components and BIE's.

2.3.1 List of General Requirements

Before a format is defined, it was important to capture the requirements for what the Data Element Metadata must be capable of supporting. In addition to the metadata requirements outlined in the UN/CEFACT CCTS and UMM, each Data Element Metadata (DEM) object should be capable of the following:

1. An XML schema and/or DTD may be derived or expressed from the DEM object, yet the DEM object must not preclude other formats of instance data from being used within an operational system in the future (such as UML, ASN1 and ASN2 etc.). Target output types include XML schema, XML DTD, HTML and binary formats such as PDF. This may also provide eForms capabilities.

2. The DEM objects shall be readable by both humans and application actors within an infrastructure and that the applications shall be able to consistently derive structure from the DEM objects. This requires a language with terse and exact parsing rules that leave no room for variance between commercial implementations of parsers or proprietary byte handling routines.

3. Binary expressions (Special syntaxes for representations such as PDF or MS Word) must have a MIME type attribute associated with them to enable application rendering within correct applications.

4. The DEM objects can explicitly point at or otherwise reference a UML or other modeling expression via a variety of protocols (examples – HTTP/S, LDAP, FTP). This places a pre-requisite for a mechanism like xlink or hypertext linking.

5. The Data Element Metadata shall have a binding to a set of RIM metadata and/or shall minimize replication of Registry meta-metadata instances except where required for data portability. This specifically refers to, but is not constrained to, using RIM Associations to express Core Components and BIE's of type "association".

6. The DEM shall not constrain the final representation in any way, yet must be capable of facilitating multiple implementation serializations (syntax bindings) as represented via the UN/CEFACT core components technical specification diagram. (NOTE: This should be termed Syntax Independent, rather than Syntax Neutral since even UML is a syntax).

7. The DEM shall be capable of conveying semantics of the core Data Dictionary Data elements in more than one language and syntax.

8. The DEM must be in a format capable of expressing multi-byte character encoding. Ideally UTF-8 and UTF-16 should be supported in order to facilitate internationalization.

9. The DEM must be capable of being transformed easily into other DEM formats (such as the UN/CEFACT ATG2 Core Components/Business Information Entities Meta-metadata format and work by the OASIS CAM and BCM groups when those groups have completed their work.)

10. The DEM must be capable of declaring semantic equivalencies to other existing metadata objects. This is a requirements based on an understanding that integration with existing systems will be essential.

11. The DEM must be capable of containing an intrinsic relationship to context declarations in order to facilitate the above requirements, possibly in addition to the registry relationships expressed within the data dictionary, ebXML RIM and ISO/EIC 11179 parts 1-5.

12. The DEM must facilitate both basic (atomic) Data Elements as well as more complex aggregates. The aggregates to be designated as UN/CEFACT aggregate core

components (ACCs) and represented as aggregate business core components using XML schema.

13. The DEM should be written in a way so programmers can write implementations, yet if the DEM model changes, the implementations will not be broken. This is referred to as forwards compatibility.

14. The DEM should have an expansion slot to accommodate more than one identifier. A mandatory UUID identifier has been required however data modelers may need to assign and use other identifiers for their own purposes. Each extra identifier must have an identifier authority associated with it.

2.3.2 Special Human Actor Requirements for Data Element Metadata serialization

Many of the requirements from this section refer to requirements read from interpreting the UN/CEFACT Modelling Methodology (UMM) N090 R12 and the CCTS methodology for discovery and use of DEM's.

1. Enable data modelers to use the data elements to build transaction sets in multiple syntaxes and representations.

2. Enable business or domain analysts to maintain a complete data dictionary and share it with multiple stakeholders.

3. Facilitate all stakeholder views necessary to facilitate harmonization of data models across multiple domains.

4. Enable key stakeholders to analyze the benefits of a registry centric concept of operations.

5. Enable programmers and systems analysts to build applications against the functionality prescribed by the registry/repository system.

9. Validate the Core Components technical specification methodology and provide feedback into that teams work. This is specifically reference able to the requirement of the CCTS team to have an independent implementation validation done.

3.0 Analyzing the Data Element Models

The UN/CEFACT Core Component Technical Specification documents requirements for recognition, development, storage, retrieval and use of data element metadata (DEM). CCTS is one of several works to examine DEM in the absence of instances.

The logical model contained herein for expression of instances of Data Element Metadata is derived from reconciliation of various models. Those works include the UN/CEFACT Core Components Technical Specification version 2.0, ISO/IEC 11179 (various works) and the ebXML Registry Information Model v 2.5.

Each of these models is examined in greater detail below.

3.1 UN/CEFACT Core Component and Business Information Entity model

Below is the model for UN/CEFACT Core Components from the version 2.0 technical specification.

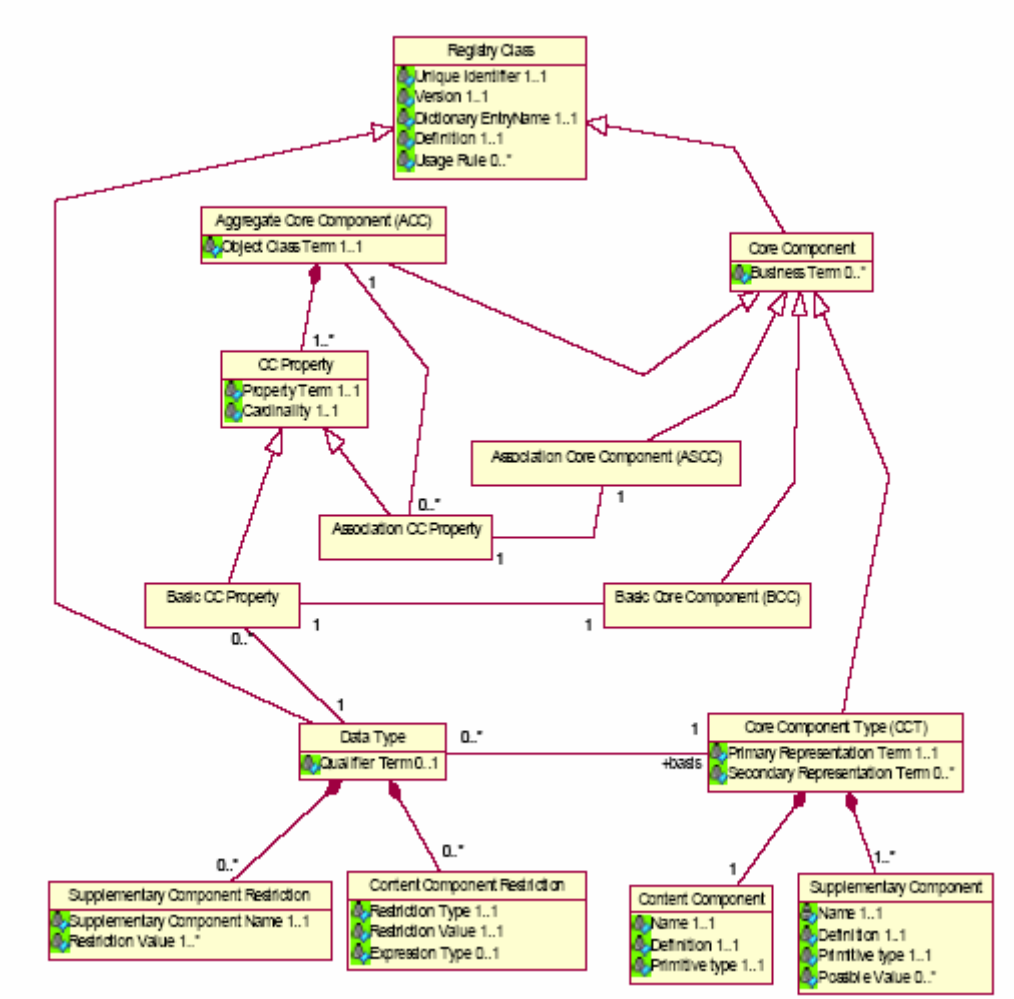


Figure 2 – Core Component model and relationships from UN/CEFACT CCTS Specification v 2.0

The UN/CEFACT Core Components Technical Specification version 2.0 contains a logical data model for a core component, albeit neutral to naming and design rules.

3.2 ISO/IEC 11179 - 2002

While not part of the normative requirements for this work, the ISO/IEC 11179 data element work is extremely helpful to examine. The concepts encapsulated within the logical model for data

element metadata have been abstracted to a higher level than several of the CCTS and ebXML RIM constructs. The 11179 model was also developed in a void from relying on an ebXML registry therefore the concepts derived from it were very useful to the work hereinafter.

	Metadata Attribute Name	Application System	
	PV End Date (each PV)	(Not applicable)	
	Value Domain (VD) Context	Registry	
	VD Entry Name	ISO English-Language Country Short Name	
	VD Definition	All short, ISO-recognized English-language names of all countries.	
	VD Description	(Not applicable)	
	VD Entry Identifier	{RAI} 5678:1	
	Datatype	CHARACTER VARYING	
	Datatype Schema/Source	ANSI ISO SQL	
	Maximum Characters	44	
	Format	(Not applicable)	
	Unit of Measure	(Not applicable)	
	Precision	(Not applicable)	
	VD Origin	ISO 3166-1:1997	
	VD Explanatory Comment	The value domain includes only the subset of names that designate countries; it does not include names of territories.	
3	Representation Class Attributes		
	Representation Class	Name	
	Representation Class Qualifier	Short	
4	Data Element Name and Identifier		
	DE Name Context	Registry	Facility Data System
	DE Name	Country Mailing Address Name	Country.Mailing_Address.Name
	DE Entry Identifier	{RAI} 5394:1	
5	Other Data Element Attributes		
	DE Example	Denmark	
	DE Origin	Application system	
	DE Comment	This data element is required for delivery of mail outside the country of origin.	
	Submitting organization	Office of Enforcement and Compliance Assurance	
	Stewardship Contact	Facility Data Systems Administrator	
6	Data Element Concept and Conceptual Domain		
	Data Element Concept (DEC) Context	Registry	
	DEC Name	Country Identifier	
	DEC Definition	An identifier for a primary geopolitical entity of the world.	
	Object Class	Country	
	Object Class Qualifier	Mailing Address	
	Property	Identifier	
	Property Qualifier	(None)	
	DEC Entry Identifier	{RAI}12468:1	
	Conceptual Domain (CD) Context	Registry	
	CD Name	Country	
	CD Definition	The primary geopolitical entities of the world.	
	CD Entry Identifier	{RAI} 2468:1	

	Metadata Attribute Name	Application System
	CD Origin	ISO 3166:1
	Value Meaning (for each VM)	The primary geopolitical entity known as <China>
	VM Begin Date (for each VM)	19970110
	VM End Date (for each VM)	(Not applicable)
	VM Identifier (for each VM)	<Assigned by system as 1001…1230: one to each VM>
7	Classification Type Examples	Classification Values for Classification Type
	Keyword	Country, Address, Mailing
	Group	Mailing Address
	Object	Address, Country
	Layer of Abstraction Type	Specialization
8	Registration and Administrative Status	
	DE Registration Status	Recorded
	DE Administrative Status	In Quality Review
	VD Registration Status	Standard
	VD Administrative Status	Final
	DEC Registration Status	Recorded
	DEC Administrative Status	In Quality Review
	CD Registration Status	Standard
	CD Administrative Status	Final

Figure 4.2 – ISO/IEC Data Element Components

The ISO/IEC 11179 Part 3 also defines types of Data Element Metadata and the relationship between administered components. The concepts are similar to both the CCTS and eb RIM work however they were a good starting point to define a serialization format for core components and business information entities.

The main concepts of administered items are depicted below:

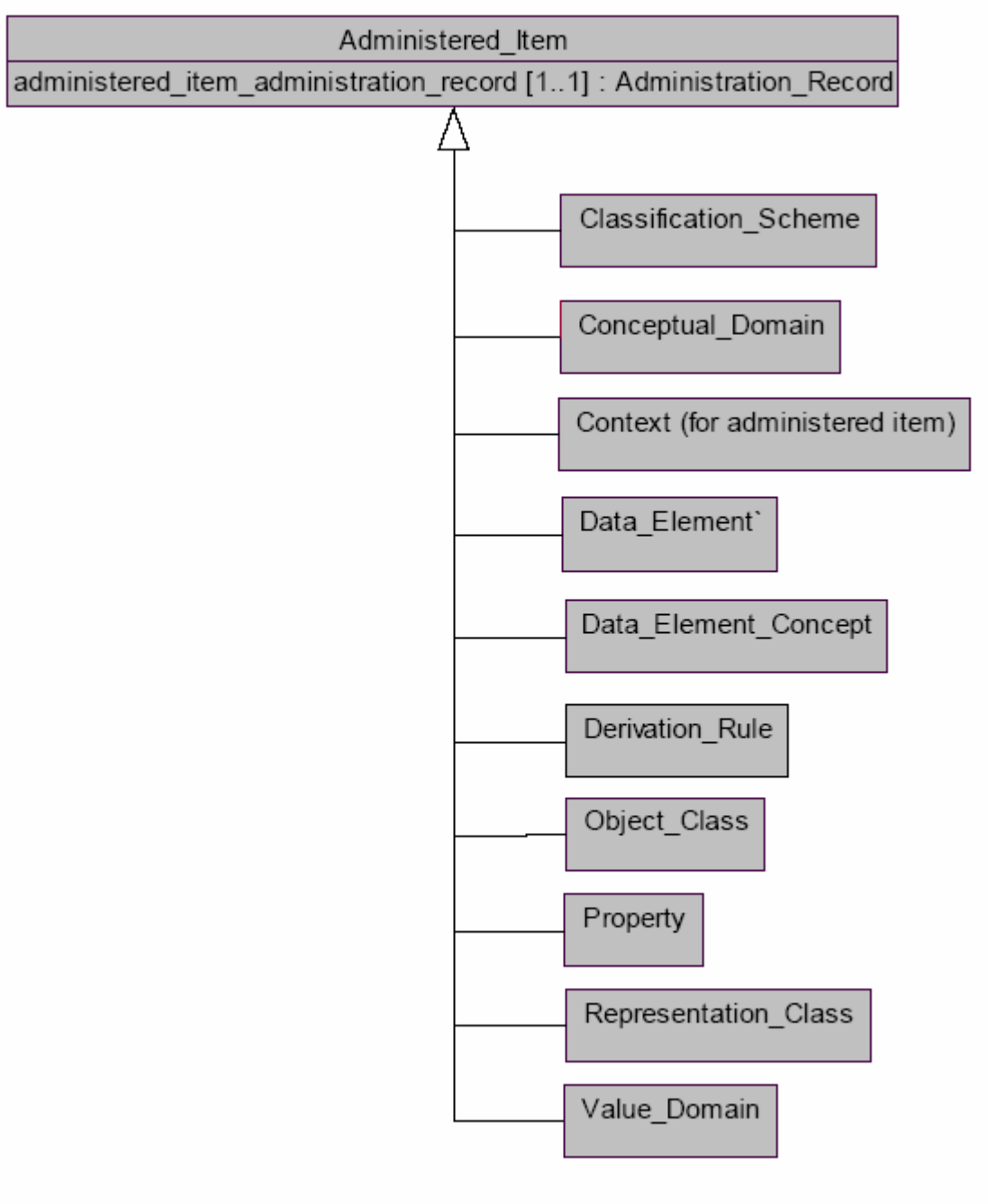


Figure 4.2.2 – types of administered items (courtesy of ISO – all rights reserved)

Several of the concepts from the figure above can be reconciled with either the eb RIM or the CCTS.

3.2.1 Classification Scheme and Concept Domain

The classification scheme and concept domain are likely best suited for representation by the eb XML RIM. As long as a core component and/or business information entity has a notion of it's home registry, a user can query that implementation for further data on how it is classified.

Because users are not likely to need this information other than to initially locate the correct data element metadata, it is best suited for representation within the registry's classification schemes.

Having the Concept Domain information present within a serialization may be a good mechanism to provide a clue to its purpose, origins and semantics.

3.2.2 Data Element, Data Element Concept and Object class

These are viewed as all being properties of data element metadata. Because each of these items are "asserted" by an actor assuming the roles of either a data element steward, submitting organization or responsible organization, all three of these items should be contained under a higher level container called "Properties" (see below).

3.2.3 Properties

Properties by themselves can all be represented within the sub-component of the instance of data element metadata. Properties can change based on the viewpoint of the beholder, therefore it is logical to assume that an attribute be assigned to each set of properties that attributes the assertion to a specific organization.

3.2.4 Representation Class, Context, Value Domain and Derivation Rules

All of these aspects of a Data Element can be reconciled within a Representation branch. This is essential since a representation is dependent on the Context and the Derivation Rules define the rules for constraining the data element metadata based on the context declaration.

The Context declaration should be defined as a standalone registry artifact and not included inline with the core components and business information entities. A business information entity is a core component, further constrained by a context declaration. The context mechanism includes unique context values for up to 8 approved context categories.

The context declaration mechanism should be kept as a separate object and each set of unique context values should be declared and given a context unique key. This key should be in the format of a DCE 128 bit algorithmic generated UUID.

The representation sub fragment of a core component should contain the UUID one or more context declarations that are applicable.

3.3 ebXML Registry Information Model

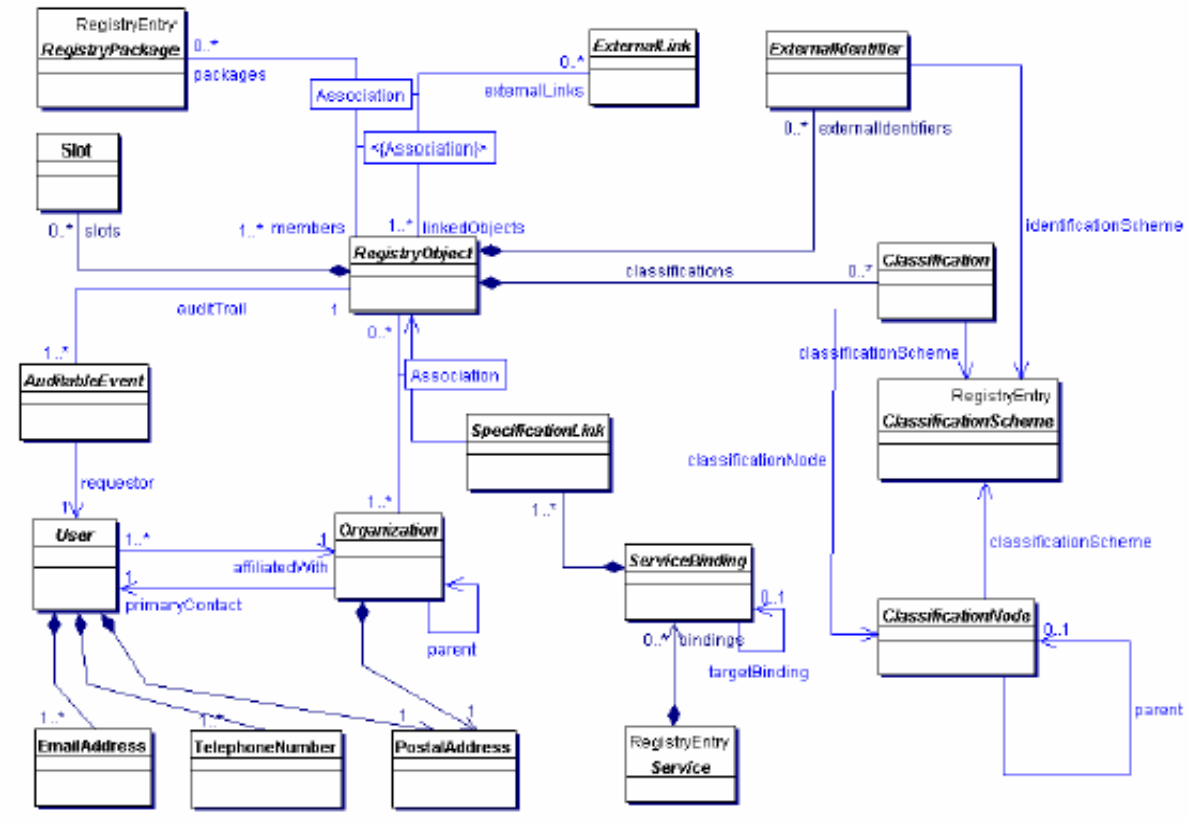


Figure 4.3 – ebXML Registry Information Model v 2.5

There is a large degree of overlap with the ISO, ebXML RIM and CCTS data models. All of these models have an identifier, versions, common name and associations with the authority that is responsible.

It is important that if a Core Component leaves the registry environment, all or some of this metadata may have to be available to the actors using it, therefore, a flexible mechanism to express some or part of all the registry metadata within the serialization of the core component is necessary. Because the Registry allows arbitrary user defined metadata about each registry object, the serialization must be equally capable of fulfilling the demands of the users.

Person's reviewing this work may also wisht o examine the ebXML Registry Information Model inheritance model for further depth of understanding.

4.0 Model Reconciliation and Proposed Solution

For the Core Components to be stored/managed in an ebXML Registry/Repository system, there is need for alignment between the CCTS properties and the ebXML Registry metamodel. Both seem to be derived from ISO/IEC 11179-*, yet the ISO/IEC standard adds another layer of complexity to creating a prototype implementation.

The first recommendation is to align the terminology used to describe certain terms.

4.1 Model Components

CCTS	ISO/IEC 11179	ebXML RIM	Proposed class of attribute for serialization
Dictionary Entry Name	Data Element Entry Name	Registry Object Name	Identifier
Definition	**Definition**		Description
Version		MajorVersion; MinorVersion	Property
n/a		Expiration date	Property
Varies		Classification Node(s)	Property
Primitive Type Minimum length (from data type)			Property
Primitive Type Maximum length (from data type)			Property
Primary Representation; Secondary Representation; Expression Type (from Data Types)…			Property
[not contextually specific until BIE]		Classification Node(s) (RIM 2.5 section 9.4	Property
Representation Term		n/a	Representation
Unique Identifier		Uuid	Identifier
Registrar, Registration Authority, **Submitting Organization**	Responsible Organization; **Submiting Organization**	Responsible Organization; **Submitting Organization**	Property
[handled by ebXML RIM]	tba	Status	Property
Business Term			Property
Dictionary name?			Identifier

Figure 5.1 – Reconciliation of the various attributes

The proposed solution is to fit the entire set of data element attributes grouped together into 4 classes bound by the main Data Element class in the model. Each mandatory attribute of a specific data element will be sorted according to Figure 5.1

The ebXML Registry Information Model makes the attribute “Status” mandatory. Since this may be primarily used to machine access, there may be a secondary of separate “Status” asserted by one or more organizations who use the DEM. It is not mandatory that these two Status attributes be synchronized since on can be retrieved programmatically from the Registry and the other one can be read from the instance; but it is a recommendation that the RIM status can also be accessed via the DEM instance serialization. Further reasons are set forth in section 5.12.

4.2 Basic UML Model of Core Component/BIE Serialization

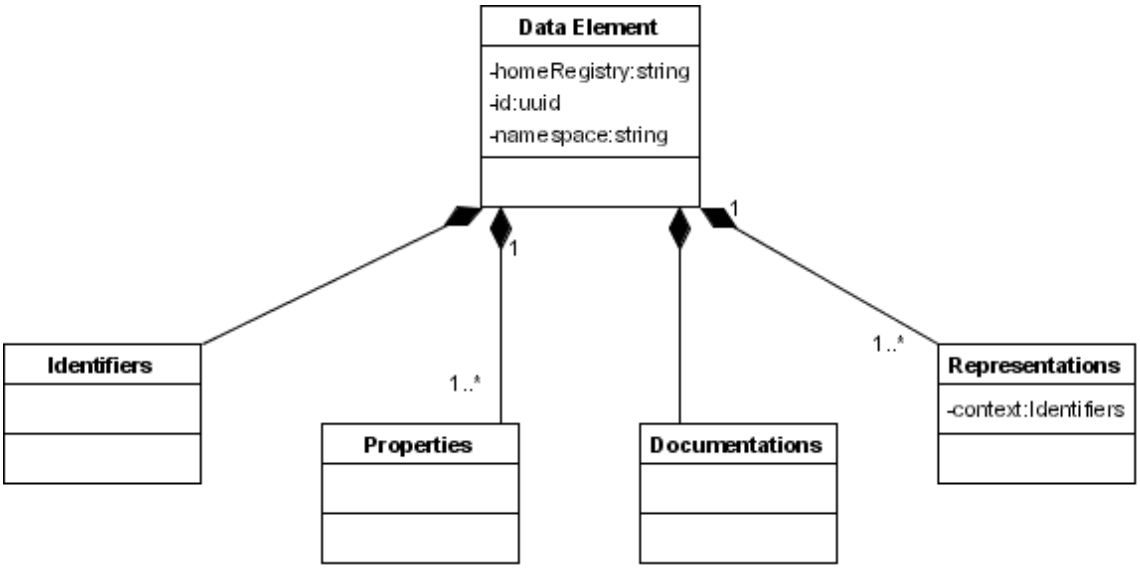


Figure 4.2 – UML expression of Core Component Serialization

The logical breakdown of a serialization of a Core Component or BIE into this format is important to understand. This model is abstracted to a higher level than the CCTS uses.

4.3 Data Element element

The <DataElement> element is the top level container of the model for serialization. The Data Element element has four child elements – *Identifiers, Properties, Documentations and Representations.*

The Data Element element also has three attributes – homeRegistryURL, id and namespace (optional).

4.3.1 Home Registry URL Attribute

The attribute homeRegistryURL is the string that resolves to the home registry of the core component or BIE. This is the base URL only and does not need to be the string that may be used to invoke a request for this specific registry object (as definable via the Registry Services Specification v 2.5 – see "HTTP binding").

The homeRegistryURL must have exactly one occurrence. The primary purpose is to allow users of the Core Component or BIE to know which specific registry they may use to retrieve additional information on this registry object (including items like associations and classifications). The secondary purpose is to fulfill the requirement for users to provide feedback and possible change requests to the Core Component or locate the *Responsible Organization* in order to clarify details about the core component or BIE, a methodology outlined within the UN/CEFACT CCTS v 2.5 and UMM.

4.3.2 Id (Identifier) Attribute

The id attribute is the universally unique identifier used to positively identify the core component across a registry federation. The attribute must be in the UUDI format specified by the ebXML Registry (the DCE 128 bit format).

> **Example:** urn:uuid:6e101f7d-3976-3d3e-5b27-095949525421

Any registry that accepts a core component must use the same UUID as is expressed internally within the core component's DataElement id attribute.

4.3.3 Namespace Attribute

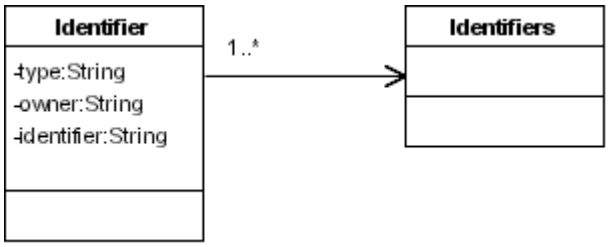
The namespace attribute is optional. It is used to declare the namespace for the core component. Namespace attribute value must be unique in order to satisfy the requirements of parsing namespace qualified elements. Accordingly, the use of URL's is recommended for namespace values.

4.4 Identifiers

Identifiers are an important aspect of data element metadata. An identifier may be the primary means for an agency to identify a certain piece of metadata. Those who are attempting reconciliation of multiple data elements from various vocabularies have a requirement to place their own identifier on a core component or BIE "owned" by another agency, possibly alongside the identifier assigned by that agency.

Another use case many be that someone using the OASIS Content Assembly Mechanism (CAM) or Universal Data Element Framework (UDEF) may need to use their own identifier within an existing core component or BIE.

The logical model (expressed in UML) is as follows:



4.4.1 Type attribute

The type of identifier being asserted. Is it a Unique identifier, a CAM identifier or some other type of identifier. The data-type is String in order to allow great flexibility.

Implementation Note: When designing an API to a Core Component or BIE, handler code can be written to supplement the parsing and grab the identifier needed by first recognizing the correct "type" attribute.

```
// using JDOM – http://www.jdom.org
Iterator idIterator = identifiers.getChildren("Identifier");
String idVariable = null;

while (idIterator.hasNext()) {
      Element id = (Element)idIterator.next();
      String type = id.getAttributeValue("type");
      if (type != null) {
            if (type.equals("neededType")) idVariable = id.getText();
      }
}
```

4.4.2 Owner Attribute

The identifier (perhaps a URI?) of the Agency or Responsible Organization who assigned the Identifier. The data type has been left as text in order to allow flexibility and extensibility. In the future, an enumerated list of values may be possible if consensus is reached on permissible values.

As with the example above, this can be also used to help identify the correct identifier you wish to retrieve and use.

4.4.3 Identifier Attribute

The actual value of the identifier. There is no requirement to have the identifier be universally unique since the owner and type can be used to qualify the value.

Readers should note that this identifier "grab bag" is in addition to a Universally Unique Identifier (UUID) contained as an attribute the top level element "DataElement".

4.4.4 Cardinality of the Identifier Element

Due to many agencies assigning different identifiers to the same data element and as a mechanism for reconciliation of multiple data elements, the Identifiers element must be able to contain multiple identifiers, asserted by multiple agencies. There is an implicit requirement to trace an identifier assignment back to the agency who made it. Therefore the Identifiers element is really equivalent to a grab bag of identifiers and agencies who assert them.

4.4.5 Sample XML Representation of the Identifiers Element

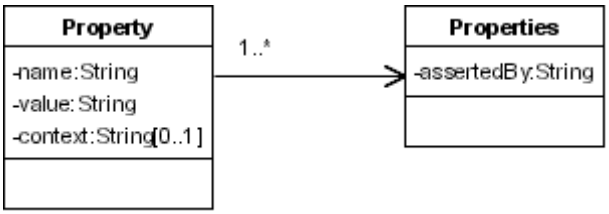
```
<Identifiers>
  <Identifier type=”myIdentifier” owner=”me” identifier=”123456”/>
  <Identifier type=”CAM”
              owner=www.oasis-open.org/committees/CAM
              identifier=”DRRW01254” />
    <Identifier type=”UDEF”
              owner=http://www.udef.org
              value=” q.3_1.1.10.10” />
      ...
</Identifiers>
```

See the schema Appendix “” for more detail.

4.5 Properties

Because properties are asserted by an agency, as with an Identifier, there must be a mechanism to trace the property assertion back to the agency who made it. Since Properties is a collection of properties, the properties branch of the expression should be identified with the agency that asserted it. Expressing the asserting agency as an attribute will help those parsing large Core Components or BIE’s to avoid others’ assertions and retrieve only those they need.

The logical model (expressed in UML) is as follows:



4.5.1 Asserted By attribute

The assertedBy attribute attributes the declaration of the set of properties to a specific organization or entity. The datatype has been set initially to string. There is a dependency that the values of the assertedBy attribute should be unique to avoid clashes, therefore the use of URL’s is highly recommended as the attribute value.

4.5.2 Property element

The Property element is the container for a set of properties asserted by an organization. The format is set to be extensible since there is great disparity between the requirements of individual organizations on what specific properties must be kept for a data element instance. Both the UN/CEFACT CCTS and ISO TC’s keep different list for their data.

The property element has three attributes – name, value and context (optional).

4.5.3 Name Attribute

The name attribute value should be a string to represent the name of the property. This helps qualify the value attribute. If, for example, an organization needs to keep information about the major. Minor and incremental versions of a core component, they would use the name attribute to set the name to qualify the attribute value, potentially using three names – *majorVersion, minorVersion* and *IncrementalVersion*.

There is no enumerated list of values for the Name attribute at this time. It is possible to build an enumerated set of values for the name attribute based on current mandatory and optional attributes used both within the UN/CEFACT CCTS v 2.0 and ebXML RIM v 2.5 specifications, however if those specifications advanced and added new values, it may render this work deprecated. It also would unnecessarily constrain users of the CCTS and Registry works from extending the base set of enumerated values to meet their requirements and also may hinder forwards and backwards compatibility. It is therefore recommended to keep the data type string.

There is a requirement that the name attributes be unique. Change requests to add new name values have to be carefully scrutinized by the organization responsible for ensuring the registry object is conforming to usability requirements.

4.5.4 Value Attribute

The value attribute contains the value for the named property. The value mechanism is of data type string. There is currently no mechanism to conditionally constrain the value attribute values to enumerated lists depending on what name attribute is used within the current version of XML Schema.

4.5.5 Context attribute

The context attribute is optional and declares that this property element declaration is true ONLY if used within a specific context. This is a requirement to guide data modelers in refining core components based on their context of usage as an interim step to creating a fully context qualified business information entity.

The context attribute data type is set to String. It is highly recommended that a UUIID is used to declare a specific context. A specific context may be made up of between one and eight context category declarations (one declaration per category).

For more information on context declaration and the assertions of logical “ands” and “ors”, please see section 6.0 below.

4.5.6 Notes regarding potential overlap with the ebXML RIM

The example below uses some of the same information that is available within the ebXML Registry Information Model. While this may be redundant if a registry is available, it does help mitigate the impact of users who need to work with the core components in environments where ebXML Registry access cannot be guaranteed.

Some aspects of the context mechanism are mapped directly to the ebXML RIM instance data. The hierarchic relationships and temporal application of context declarations shall be represented using classification schemes.

4.5.7 Sample XML Fragment for Properties

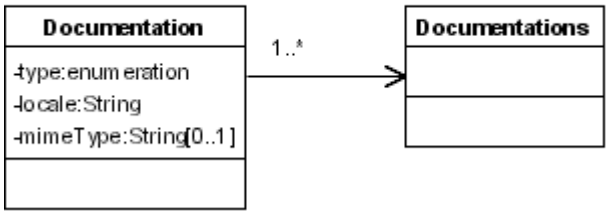
Below is an XML sample for a <Properties> branch.

```
<Properties assertedBy="Canadian Public Safety Information Network">
      <Property name="version.major" value="1" />
      <Property name="version.minor" value="0" />
      <Property name="version.incremental" value="0" />
      <Property name="registration.status" value="APPROVED" />
      <Property name="domain" value="dDateValue" />
      <Property name=”example” value=”no matter what I put here and in
the name attribute, it will not break implementations.  There is a
problem with unique values for the name attribute however.  The
responsibility to avoid duplicity of name attribute values lies on the
Responsible Organization or other registry administrator” />
      <Property name=”context” value=”IJI Context” />
      <Property name="topic" value="Person" />
      <Property name="familiar.name"
                  value="Person.Gender.Identifier"
                  context="used in English speaking regions to declare
that this is the preferred name to use when addressing a subject" />
      <Property name="synonyms"
                  value="Animal.Gender.Identifier"
                  context="a fully qualified name to be used in ISO
11179 data modeling work." />
   </Properties>
```

4.6 Documentations

The Documentations element is the highest level container for declarations of additional documentation about a core component and/or it’s business information entities. As with most efforts to reconcile metadata on a global basis, Documentations must be capable of being localized in many languages.

The logical model for documentations is as follows:



4.6.1 Documentation Element

Within each Documentations element, there may be one or more Documentation elements. Each Documentation element may have up to three attributes – type, locale and mimeType (optional – if known).

4.6.2 Type Attribute

The Type attribute is the type of documentation and must conform to a choice from an enumerated list of values. The enumerated list of values is of data type String[] (an array or list of type String).

The documentation is present based on a requirement to provide extra information about a specific core component and/or its associated business information entities to human actors to help them clarify semantics or other details.

The enumerated list of values for the type attribute is:

```
( comment|note|instruction|other )
```

4.6.2.1 Comment value

The comment value indicates to the actor reviewing the artifact that the documentation is of type *comment*. A comment is a non-normative string present in order to declare additional information about a specific core component.

4.6.2.2 Note value

The note value indicated to the actor reviewing the artifact that the documentation is of type *note*. A note is of higher priority than comment and suggests to the actor that it is highly recommended to read the note. It may contain information about future deprecation or stability.

4.6.2.3 Instruction value

The Instruction value indicated to the actor reviewing the artifact that the documentation is of type *instruction*. This is considered a normative in nature. It carries more weight than either the Comment or Note values

4.6.2.3 Other value

This is left for items that do not correspond to the other three choices. The datatype is string.

4.6.3 Locale attribute

The locale attribute is used to declare that a specific documentation is linked to a locale. It is strongly recommended that the code list of permissible values for the locale attribute are the ISO-3166-2 country and region codes. This will facilitate global understanding of the locale value.

4.6.4 mimeType Attribute

The mimeType attribute is optional and is used as a mechanism to flag applications of the mimeType of the Documentation. This is important since CDATA sections are permitted within the Documentation element and a mechanism must be present to aid application correlation.

4.6.5 Example XML Serialization of Documentations Element

The Documentations branch of the serialization may look as follows

```
<Documentations>
    <Documentation type="comment|note|instruction|other"
                   locale="en CA"
                   mimeType="text/html">
                   <![CDATA[<html><body>Element Approved but further
                   research needed for values</body></html>]]>
    </Documentation>
    <Documentation type="comment|note|instruction|other"
                   locale="fr CA"
                   mimeType="text/html">
      <![CDATA[<html><body>viva la difference!</body></html>]]>
    </Documentation>
  <Documentations>
```

4.7 Representations

The first three branches of the serialization solution (Identifiers, Properties and Documentations) all deal with aspects of core components. The Representations branch of the model declares the BIE's and contains details of specific constraints, the link to the context of usage and a serialization of the data elements according to context. The exact mime type and representation for a specific context is dependent upon the contextual requirements for deployment. For example, if the Systems Capability context is set to "human reading a form", then a binary chunk of data representing a PDF eForm may be present. If the systems capability context is set to "XML schema", an XML schema fragment may be there (perhaps in RelaxNG format, being the superior schema format).

4.7.1 Basic UML Model of Representations

The base model for the representations branch is as follows

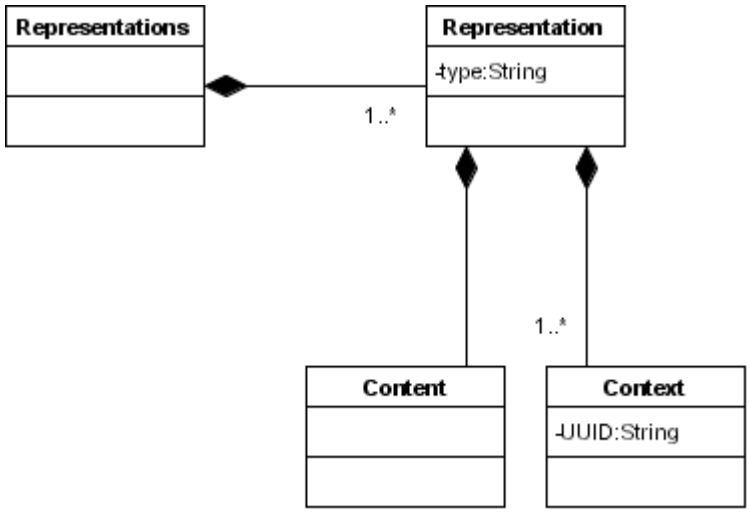


Figure 4.7 – UML of Representations (BIE) branch

A unique branch of the serialization may have to be present for details on how to represent each BIE based on the CC it is based on. The solution hereinafter places the Business Information Entities (BIE's) inline within a Core component. This is purely an arbitrary choice since several mechanisms exist to link to remote (extrinsic) content.

> **Implementation Note:** If the BIE is an XML schema fragment, expressed in the W3C XML Schema format, the xs:import element can be used to reference remote content. If the Core Component is within an ebXML Registry Repository, the ebXML Registry has both an associations mechanism and also a URL mechanism that makes use of he HTTP binding to the registry interface to reference content.

The inline BIE's are housed within the Representation element. The Representation element may contain multiple BIE's – each one conforming to a unique context declaration (linked via the UUID mechanism).

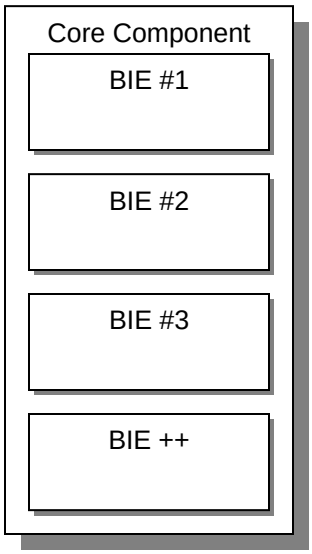


Figure 4.8 – BIE's serialized inline inside a core component

Remembering that BIE's must be able to be expressed in many different formats, this branch must be capable of relaying both text and binary data. It also has to be reconciled with each unique set of contexts. There are currently 8 contexts declared by the CCTS v 2.0.

It is tempting to form the Representations branch as follows:

```
<Representations>
   <Representation context=″UUID_of_set_of_contexts″>
   …text or binary data here…
   </Representation>
```

However, it is also anticipated that one representation may be used for more than one set of contexts. This includes duplicity for both logical "ANDS" and "ORS". It is also anticipated that individuals may use remote mechanisms like CAM to declare their assemblies or representations based on the contexts. Furthermore, users may embrace data element reconciliation methodologies like UDEF and require a UDEF linking mechanism.

A linking mechanism must be present to declare that two or more contexts can use the same BIE.

```
<Representations>
   <Representation>
     <Context contextID=″uuid_of_set_of_contexts″ owner=″″/>
     <Context contextID=″
uuid_of_another_set_of_contexts_that_use_this_BIE″
      owner=″″ />

     <Content><CDATA[>  ...content (BIE) goes here...  </]></Content>
   </Representation>
```

5.0 Mapping of specific Core Components terms to model

The following table maps elements and attributes of the core components model, as derived from the UN/CefAct CCTS v 2.0 technical specification, to the constructs of the serialization solution proposed herein.

The following represents a list of properties and attributes deemed necessary for each DEM. The list is not necessarily inclusive as actors using the CCTS methodology may also need to deploy additional properties to allow CCTS to bind to UDEF, CAM or other related work.

Property or Attribute name	Description	Comments	Category of aspect
UUID	Universally unique identifier. The registry can provide a UUID in the form of a DCE 128 bit algorithm generated from a seed value.	Would recommend re-using the same format for the core components UUID but supplementing it with a property value of the URL of the registry that is the Data Stewards home Registry.	Identifier
Version	The version of a DEM, according to the registry.	Would recommend breaking this into version.major; version.minor and version.incremental to further control access to correct versions. May need to sync this up with the Data Stewards versioning and having a more robust versioning capability may facilitate mapping to other models.	Property
Dictionary Entry Name	The English (ISO EN-uk) language entry name, using the period concatenation of qualifier and representation terms.	Must keep. Possible to expand to support other languages that English?	Identifier
Definition	Semantics	Definition is only high level. For Core Components, definition is exclusive of any specific context(s). For BIE's a way to reference the context declaration that was used to help constraint the definition is imperative.	Property or Documentation

Business Term		Needs context to define.	Property or Documentation
Property	Unabounded instances of properties associated to this core component	The representation needs to account for a property name, property value and cardinality. Perhaps an additional value for a qualifier may also be of use for enumerated lists of values as a guide to qualify the value.	Property
Associations	How one DEM is associated with another	Probably best handled via the registry association mechanism but will need to develop a clearer understanding of how specific instances may be represented.	RIM
Core Component Type	Described additional properties about the core component	Perhaps these are best represented under the "properties" of the core component.	Representation
Core Component Restriction	Constraints that affect representations of instances of the DEM's. Likely dependent upon context declarations.	This is ideally expressed in the "Representations" are of a core component.	Representation
Supplementary Component Restriction	As above with Core Component Restriction, these are further constraints.	An aspect of representation of instances.	Representation
Supplementary Component: Possible Values	An enumerated set of values permissible for this DEM	An aspect of representation. Include cardinality, data type and other constraints for structure.	Representation
Supplementary Component: Primitive Type	Primitive data type	Can be constrained by XML Schema as part of the representation	Representation

6.0 Context Declaration Mechanism

Section 9.4 of the ebXML Registry Information Model discusses using the classification schemas and nodes as a mechanism to express contextual classifications of registry objects. This is the methodology we also recommend based on the ease of which this mechanism can express multiple contextual classifications and may be extended to meet future or other requirements in this area.

6.1 Context – relationship to Core Components and BIE's

A basic core component is a highly abstract concept equivalent to a single data element. As such, it is not suited for direct use to constrain instance data. It must be subject to further constraints based on the context of its' intended usage. When a Core Component Data Element is constrained within a specific set of context category values, it becomes a Business Information Entity or BIE.

The following UML model reflects the relationships between the core components, business information entities, registry and context assertions.

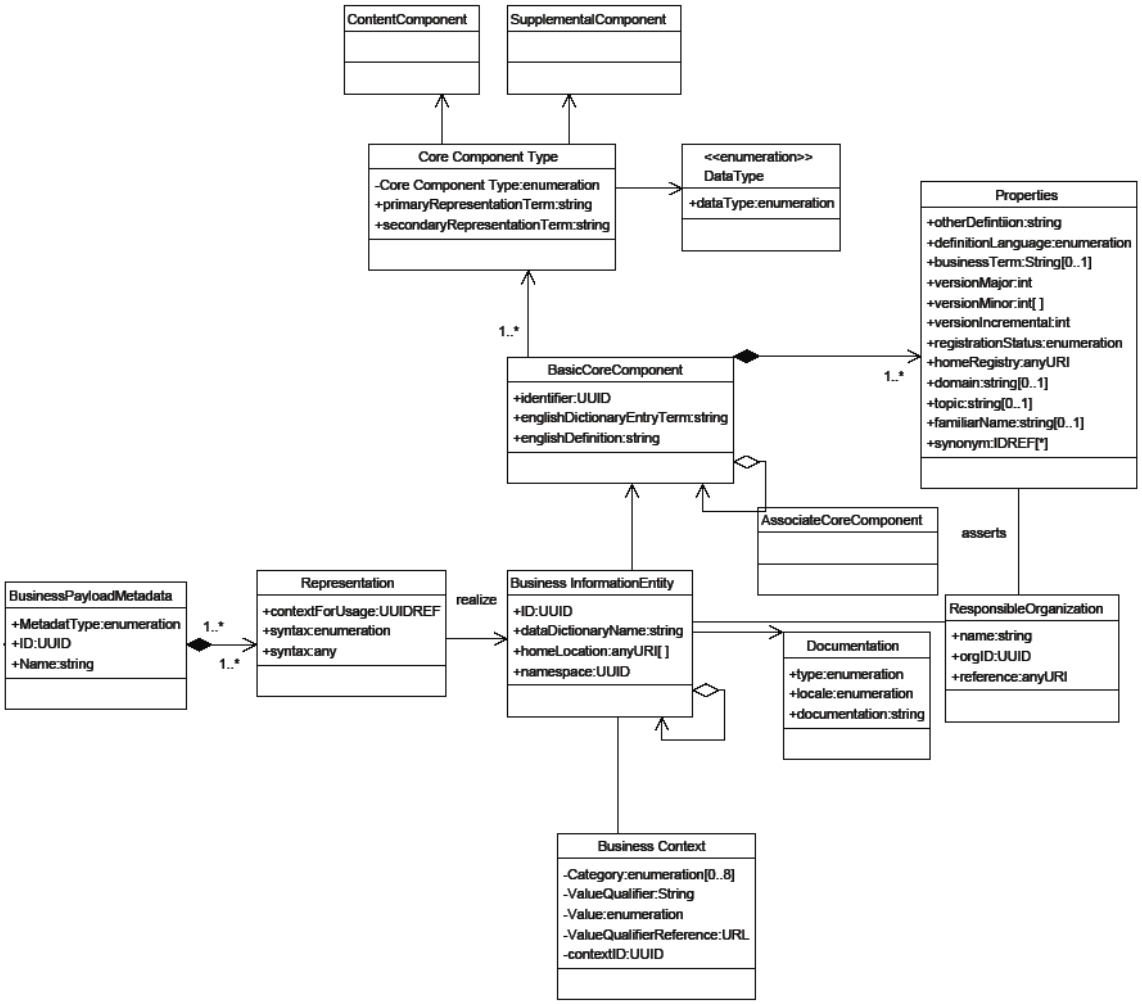


Figure 6.1 – relationships between core components, business information entities, context, registry and repository.

The UN/CEFACT CCTS v 2.0, ISO 11179, ebXML Registry/Repository and UMM all adhere to the context methodology. During the design phase, business modelers use the UN/CEFACT Modeling Methodology to capture the context of a data elements usage.

Implementation Note: A problem does exist however with respect to registry classification scheme bloating. If all the CCTS context classifications are used for the purpose of aiding modelers in locating contextually specific business information entities, the registry classification scheme would be unmanageably large.

More about the Bloating Issue

The latter Implementation Note is worth a closer look. For example, if you chose to express just 4 context categories and had 50 values for each possible context, you would have to create 50^4 classification schemes to express one specific order of classification. In reality, the numbers are much larger.

Context	Classification scheme	Number of values (approx.)
Geopolitical	ISO 3166-2 ISO 639	1,650 (165 countries * 10 regions) * the number of languages
Industry Classification	NAICS	3,950
Business Process	UN/CEFACT Catalog of common business processes	50 (unknown at this time)
Supporting Process	UN/CEFACT Catalog of common business processes	50 (unknown at this time)
Official constraints	Unknown (United Nations + each nations legislation (and United Nations)	5000
Role	Depends on processes.	25 (guess)
Systems Capabilities	Unknown	25 (guess)
Product Classification	UN/SPSC	3,250 +

If you account for every possible combination, this may not work very well or take a long time to implement.

CCTS Model

A more exact model for the declaration of context can be derived from the UN/CEFACT Core Components Technical Specification v 2.0 (CCTS). Within that document, a model exists for declaring sets of contexts. The model is shown below.

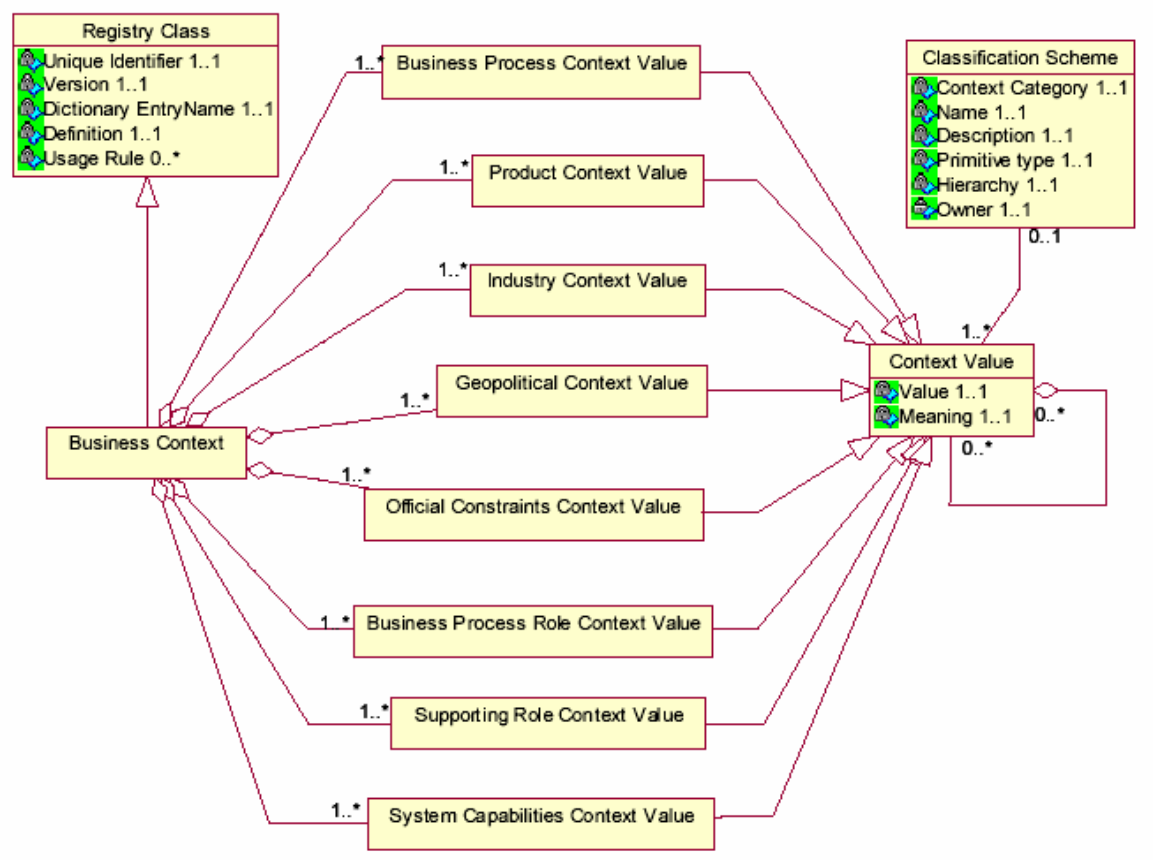


Figure 8 – Core Component model for Context from UN/CEFACT CCTS Specification v 2.0

Each individual context has three parts to it. A context category identifier, a qualifier for a list of code values that are acceptable to express the values for that context, and a value or set of values specific to an individual context. There are hierarchic considerations that may be best addressed using the registries classification schemes and the owner of those classification nodes is handled natively in the registry (all regions of a registry information model extend from the administration region of that registry).

It may also be a good idea to include the identifier for an agency or place where a stakeholder could retrieve additional information about a specific context coded value list.

6.2 Requirements for Context

The list of eight context categories may not be complete and care should be taken to build a Data Element in a format that will not break should another context category be added in the future.

There must be a mechanism for a Business Information Entity to declare that it is to be used for a certain set of context category declarations. A Business Information Entity must also know the core component it was derived from (visibility of core component from instance BIE).

One aspect of context that is relatively ignored is the application of context at both design time and during the modeling/design phase. The dependencies between context categories are not well understood and may have serious implications for implementers. An example is how a geo-

political classification may have to be the first context discovered since it could dictate language requirements for understanding the following contexts.

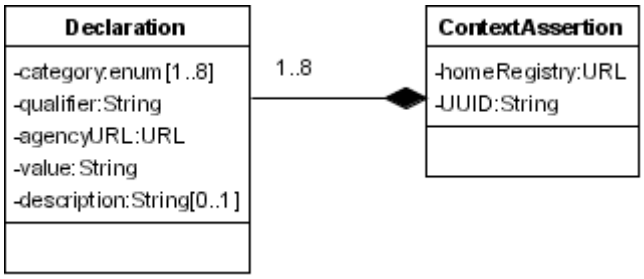
The hierarchic and temporal aspects of context are not covered herein.

6.3 Plurality of Contexts

Re-use of core components and BIE's is a high priority amongst the core components requirements. Accordingly, there should also be a way to declare that multiple contexts should use the same BIE (logical "ands") and a way for more than one BIE to be used within many contexts (logical "ors" from the BIE viewpoint).

6.4 UML model for Context Declaration Relationships

The UML following UML model is based on the CCTS v 2.0 specification and ebXML Registry/Repository Specifications.



6.4.1 The ContextAssertion Element

The ContextAssertion is the top level element container for a specific context declaration. It contains a set of up to eight context declarations (each of which uses the Declaration element). Each Declaration Element must contain a minimal set of data and have a unique context category attribute value. The enumerated list of permissible values is in alignment with the UN/CEFACT CCTS v 2.0 Technical Specification.

For more information on permissible values for context categories, please see 6.4.5 The Category Attribute below.

6.4.2 The Home Registry Attribute

The homeRegistry attribute is mandatory and contains the URL of the homeRegistry for the specific context assertion. The datatype is String. This is essential to fulfill the requirements of

data modelers who may need to request additional information (including classification schemes that declare additional aspects of hierarchic relationships of context and/or temporal ordering of application of contexts) and a way to locate the administrator for the context assertion to facilitate the UN/CEFACT CCTS methodology for change requests or other aspects of management.

6.4.3 The UUID Attribute

The UUID attribute value is a universally unique identifier in the same format that the ebXML Registry-repository uses. This shall be in the DCE 128 bit format.

Example: urn:uuid:6e101f7d-3976-3d3e-5b27-095949525421

When context assertions are submitted to multiple registries, they need to use the same UUID across all registries.

Implementation Note: The use of the UUID will make it easier for implementers to programmatically access the correct business information entity. Rather than having to match up to eight context category conditions in order to find and reuse a specific BIE derived from a core component, they may simply use the UUID as a "key" to match up and locate the correct BIE. This will save considerable iterations of string and text matching.

If each user attempting to find a specific BIE had to nest a series of "if" statements and search across eight contexts, the code would appear something like this:

```
for (int i = 0; i < length(contextAssertions); i++){

  if (myContextCondition1.equals(ContextAssertion.getChildren("Declaration").
      getAttributeValue("category")) &&
     (myContextCondition2.equals...
    // repeat 8 times – one for each context
```

This code would use an extraordinarily large amount of virtual and physical memory space during the execution.

As an alternative, matching via a UUID is relatively easy and leave a lighter footprint.

```
If (rep.getAttributeValue("type").equals(type) &&
rep.getAttributeValue("context).equals(myContextUUIDKey) {

  //do something useful here like grab the BIE
  detach(BIE);
  }
```

6.4.4 The Declaration Element

The Declaration element is the container for each context assertion. Each Declaration within one ContextAssertion Element must have a unique *category* attribute, each choice matching one of the list of enumerated values allowable for contexts.

Implementation Note: The design of the model for the context declarations assumes that it may be extended at a future date. The design goal to preserve investments in forwards compatibility resulted in a format that can allow additional values to be used later for the context categories, as declared within the category attribute (see below). If a ninth context category was used, existing implementations that are designed to consume this model will still work and newer implementations will still be able to consume older instances of context assertions.

6.4.5 The Category Attribute

The UN/CEFACT CCTS v2.0 identifies eight context categories.

- Geopolitical
- Business Process
- Supporting Business Process
- Role
- Official Constraint
- Systems Capabilities
- Product Classification
- Industry Classification

The category attribute value declares which category of context the Declaration Element is for. Within each ContextAssertion element, the values for each category attribute must be unique.

Implementation Note: There is currently no way to support expressing the requirement for unique attribute values (conditional validation) within the W3C Schema format (*.xsd). It is therefore up to implementers to be prudent when building context assertions and declarations. The problem would be that a second, duplicate value will be ignored by most parsers since the code to retrieve them will iterate through a list of Declaration elements and examine the value for the category attribute for each one. If the do-while loop type construct is used within the implementers code, it will stop as soon as it passes through the first positive match.

To demonstrate this, imagine the following set of context declarations with a duplicate category attribute:

```
<?xml version="1.0" encoding="UTF-8" ?>
<ContextAssertion homeRegistry="http://ebxml.pwgsc.gc.ca:8080"
                  UUID="urn:uuid:4a593056-3509-0766-2e7b-
4e154030423f" >
  <Declaration category="Geopolitical"
               qualifier="ISO-3166-2"
               agencyURL="http://www.iso.org"
               value="CA-ON" />

  <Declaration category="IndustryClassification"
               qualifier="NAICS-2002"
               agencyURL="http://www.naics.org"
               value="9221"
               description="Justice, Public Order, and Safety
                            Activities" />
<!—This next category of Geopolitical will be ignored by the code
   following this block→
```

```
  <Declaration category="Geopolitical"
              qualifier="ServiceOutputSyntax"
              value="xml schema xsd" />

</ContextAssertion>
```

Code would typically be written like this to access it.

```
List declarationList = rootElement.getChildren(“Declarations”);

for (int I = 0; I < declarationList.Length(); i++) {
Attribute cat =
     declarationList[i].getAttributeValue(“Category”);
   //conditional test
  if (cat.equals(“Geopolitical”) {
      /*then do something with the value here only if the test
        is positive.  The rest of the list will not be examined
        any more and the code will stop executing yielding
        funny results if you expected to find the second
        Geopolitical category attribute value.*/
  }
  exit(0);
}
```

One way to mitigate this problem would be to declare and initialize an Iterator object then use the “While-do” loop and use the hasNext() method construct to ensure the entire list has been evaluated:

```
   Iterator categoryList = this.category.iterator();
      while (repIter.hasNext())
      {
              /* Do something with the code like test for
                 conditions. Even if there is a positive match,
                 it will keep evaluating until
                 The entire list is iterated.
                 etc… */
```

Forwards Compatibility note: At some time in the future, it may be necessary to include multiple values of one context category as legitimate within one context declaration. An example of this may be to say that all French speaking geopolitical regions should use the same BIE context assertion. This may include a list of ISO 3166-2 country qualified region codes. This would substantially change the model for the context assertion.

At this time, however, the implications are not well enough understood to build this into the model.

6.4.6 The Qualifier Attribute

The qualifier attribute is used to declare the code list qualifier for a set of values. The data type is String. For example, if you are going to use ISO 3166-2 for country qualified region codes as an enumerated list of permissible values, then the qualifier attribute would reflect this:

```
qualifier="iso-3166-2"
```

6.4.7 The AgencyURL attribute

The agencyURL attribute is where you can find more information out about the agency who is the responsible organization for the qualifier attribute. The data type is String

For example, if the qualifier list is ISO 639 currency codes, the AgencyURL should point at ISO.

This is an optional attribute.

6.4.8 The Value Attribute

The value attribute declares the value for the context declaration. The data type is String.

The permissible values for the value attribute are dependent upon the qualifier value. For example, if the qualifier for the industry context is NAICS, a value of 9221 may be permissible. If the qualifier for the same context became the UN-SPSC, 9221 would no longer be a valid value.

6.5 Sample XML instance for context declaration

The context format may be expressed using XML as follows:

```
<?xml version="1.0" encoding="UTF-8" ?>
<ContextAssertion homeRegistry="http://ebxml.pwgsc.gc.ca:8080"
                  UUID="urn:uuid:4a593056-3509-0766-2e7b-4e154030423f" >
  <!—Category. Choices = ( Geopolitical |
                           OfficialConstraint |
                           Process |
                           ProcessRole |
                           SupportingRole |
                           ProductClassification |
                           IndustryClasification |
                           SystemCapability
                           )-->
  <Declaration category="Geopolitical"
               qualifier="ISO-3166-2"
               agencyURL="http://www.iso.org"
               value="CA-ON" />
  <Declaration category="IndustryClassification"
               qualifier="NAICS-2002"
               agencyURL="http://www.naics.org"
               value="9221"
               description="Justice, Public Order
                            and Safety Activities" />
  <Declaration category="SystemCapability"
               qualifier="ServiceOutputSyntax"
               value="xml schema xsd" />
</ContextAssertion>
```

This format has several design considerations that should be incorporated into any final design.

a. It's hierarchy is simple and adding additional context categories at a later date will not break any existing implementations by fault of not being able to process the existing categories.

b. It allows for multiple qualifiers for context values. It is not fixed to any one set of values.

c. It is flexible and allows context sets to be declared that are incomplete (the example has declared only 3 out of 8 context categories).

d. It is identifiable via a single string of a UUID. This makes matching up BIE's much easier and less intensive on system resources.

Implementation Note: A question of whether a null value for a context needs to be explicitly declared is outstanding. It is highly recommended that all eight context categories are declared and if there is not value, the value attribute be set to the String "noValue". This is different from a value of null which should also be explicitly declared by using the string "null".

This will likely make it easier to trap logic errors and ensure reusability of BIE's.

7.0 Sample XML Expression of Core Components and Business Information Entities.

Based on the data model and an interpretation of the ISO/IEC 11179 and ebXML Registry models, the following sample illustrates how an XML serialization of the Data element model may be specified. A schema has been developed for this model and is attached hereto as Appendix "A".

```
<?xml version="1.0" encoding="UTF-8" ?>
<DataElement
      home="http://ebxml.pwgsc.gc.ca"
      id="urn:uuid:6e60580b-4538-2615-0c2c-3e034c430445"
      xmlns="http://ns.cpsin.org/data-element/1.0"
      >
  <Identifiers>
      <Identifier type="responsibleOrgURL"
                value="http://www.scc.gc.ca/CPSIN/iji-iij/" />
      <Identifier type="ElementIdentifier" value="014" />
```

```
    <Identifier type="DataDictionaryName"
                  value="Gender.Identifier"
                  xml:lang="en-CA" />
    <Identifier type="EntityName"
                  value="Being, Gender" />
    <Identifier type=”udef” value=”gfd123RDD” />
    <Identifier type=”CAM” value=”DRRW123456” />
</Identifiers>

<Properties assertedBy="Canadian Public Safety Information Network">
    <Property name="version.major" value="1" />
    <Property name="version.minor" value="0" />
    <Property name="version.incremental" value="0" />
    <Property name="registration.status" value="APPROVED" />
    <Property name="domain" value="dDateValue" />
      <Property name=”context” value=”IJI Context” />
    <Property name="topic" value="Person" />
    <Property name="familiar.name"
                value="Person.Gender.Identifier"
                context="A numeric value corresponding to the
                         gender which a person belongs" />
      <TODO – Add “natural language name” – replaces xml:lang→
    <Property name="synonyms"
                value="Animal.Gender.Identifier"
                context="A numeric value corresponding to the
                gender which an Animal belongs" />
 </Properties>

<Documentations>
      <Documentation type="comment|note|instruction|other"
                     locale="en_CA"
                     mimeType="text/html">
    <![CDATA[<html><body>Element Approved but further
               research needed for values</body></html>]]>
      </Documentation>
      <Documentation type="comment|note|instruction|other"
                     locale="fr_CA"
                     mimeType="text/html">
    <![CDATA[<html><body>viva la difference!</body></html>]]>
    </Documentation>
<Documentations>

<Representations>
    <Representation type="http://www.w3.org/2001/XMLSchema"
       context="urn:uuid:4a593056-3509-0766-2e7b-4e154030423f">
    <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
                  targetNamespace="urn:component-foo">
    <!--schema here-->

    <xsd:element name="Sex">
      <xsd:complexType>
        <xsd:simpleContent>
          <xsd:extension base="xsd:string">
          <xsd:attribute name="value" use="required">
            <xsd:simpleType>
              <xsd:restriction base="xsd:NMTOKEN">
                <xsd:enumeration value="01 - Male"/>
```

```
                  <xsd:enumeration value="02 - Female"/>
                  <xsd:enumeration value="03 - Asexual"/>
                  <xsd:enumeration value="04 - Transgendered - in
transition"/>
                  <xsd:enumeration value="05 - Transgendered - complete
to female"/>
                  <xsd:enumeration value="06 - Transgendered - complete
to male"/>
                  <xsd:enumeration value="07 - Hemaphrodyte"/>
                  <xsd:enumeration value="08 - Unisexual Species"/>
                  <xsd:enumeration value="09 - Not applicable"/>
                  <xsd:enumeration value="10 - Other"/>
                </xsd:restriction>
              </xsd:simpleType>
            </xsd:attribute>
            <xsd:attribute name="uuid" use="fixed" >
              <xsd:simpleType>
                <xsd:restriction base="xsd:string">
                  <xsd:enumeration value="urn:uuid:6e60580b-4538-2615-
0c2c-3e034c430445" />
                </xsd:restriction>
              </xsd:simpleType>
            </xsd:attribute>
            </xsd:extension>
          </xsd:simpleContent>
        </xsd:complexType>
      </xsd:element>
      <xsd:element name="GenderIdentifier">
        <xsd:complexType/>
      </xsd:element >
      </xsd:schema>
      </Representation>

      <!--Start of another context here-->
      <Representation type="http://www.w3.org/2001/XMLSchema"
context="urn:uuid:6563671c-5008-464c-5b38-1377054b5a7a">
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
targetNamespace="urn:component-foo">
      <!--schema here-->

      <xsd:element name="SexIdentifier">
        <xsd:complexType>
          <xsd:simpleContent>
            <xsd:extension base="xsd:string">
            <xsd:attribute name="value" use="required">
              <xsd:simpleType>
                <xsd:restriction base="xsd:NMTOKEN">
                  <xsd:enumeration value="01 - Homme"/>
                  <xsd:enumeration value="02 - Femme"/>
                  <xsd:enumeration value="03 - Asexual"/>
                  <xsd:enumeration value="04 - Transgendered - dans la
transition"/>
                  <xsd:enumeration value="05 - Transgendered -
accomplissez a la femme"/>
                  <xsd:enumeration value="06 - Transgendered -
accomplissez a la homme"/>
                  <xsd:enumeration value="07 - Hemaphrodyte"/>
```

```
                  <xsd:enumeration value="08 - Esp&#233;ce
D'Unisexual"/>
                  <xsd:enumeration value="09 - Non applicable"/>
                  <xsd:enumeration value="10 - Autre"/>
                </xsd:restriction>
              </xsd:simpleType>
            </xsd:attribute>
            <xsd:attribute name="uuid" use="fixed" >
              <xsd:simpleType>
                <xsd:restriction base="xsd:string">
                  <xsd:enumeration value="urn:uuid:6e60580b-4538-2615-
0c2c-3e034c430445" />
                </xsd:restriction>
              </xsd:simpleType>
            </xsd:attribute>
            </xsd:extension>
          </xsd:simpleContent>
        </xsd:complexType>
      </xsd:element>
      <xsd:element name="GenderIdentifier">
        <xsd:complexType/>
      </xsd:element >
      </xsd:schema>
      </Representation>


  </Representations>

</DataElement>
```

Appendix “A” – XSD Schema

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Authors: Matthew MacKenzie, Duane Nickull -->


<xs:schema targetNamespace="http://ns.cpsin.org/data-element/1.0"
xmlns:de="http://ns.cpsin.org/data-element/1.0"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified" attributeFormDefault="unqualified">
  <xs:element name="DataElement">
    <xs:annotation>
      xs:documentation>Specification of DataElement.</xs:documentation>
    </xs:annotation>
    <xs:complexType>
      <xs:sequence>
      <xs:element name="Identifiers">
          <xs:annotation>
                                    <xs:documentation>Collection
element to hold 1-unbounded Indentifier instances.</xs:documentation>
                                </xs:annotation>
                                <xs:complexType>
                                      <xs:sequence>
                                            <xs:element name="Identifier"
maxOccurs="unbounded">
                                                  <xs:annotation>

      <xs:documentation>Simple type/value element representing a piece
of information which canonically identifies data
element.</xs:documentation>
                                                  </xs:annotation>
                                                  <xs:complexType>
                                                        <xs:attribute
name="type" type="de:IdentifierTypes" use="required"/>
                                                        <xs:attribute
name="value" type="xs:string" use="optional"/>
                                                  </xs:complexType>
                                            </xs:element>
                                      </xs:sequence>
                                </xs:complexType>
                          </xs:element>
                          <xs:element name="Properties" minOccurs="0">
                                <xs:annotation>
                                      <xs:documentation>Collection
element to hold 1-unbounded Property instances.</xs:documentation>
                                </xs:annotation>
                                <xs:complexType>
                                      <xs:sequence>
                                            <xs:element name="Property"
maxOccurs="unbounded">
                                                  <xs:annotation>
```

```
      <xs:documentation>Element providing metadata storage for a
DataElement.  Property types are added to the PropertyTypes simpleType
in the schema, allowing extensibility of metadata without structural
changes to the overall schema.</xs:documentation>
                                                </xs:annotation>
                                                <xs:complexType>
                                                      <xs:attribute
name="name" type="de:PropertyNames" use="required"/>
                                                      <xs:attribute
name="value" type="xs:string" use="optional"/>
                                                      <xs:attribute
name="context" type="xs:string" use="optional"/>
                                                </xs:complexType>
                                          </xs:element>
                                    </xs:sequence>
                                    <xs:attribute name="assertedBy"
type="xs:string" use="optional"/>
                              </xs:complexType>
                        </xs:element>
                        <xs:element name="Documentation" minOccurs="0">
                              <xs:annotation>
                                    <xs:documentation>Collection
element to hold documentation entries.  Entries can be differentiated
by locale, type and mimeTYpe.</xs:documentation>
                              </xs:annotation>
                              <xs:complexType>
                                    <xs:sequence>
                                          <xs:element name="Entry"
maxOccurs="unbounded">
                                                <xs:annotation>

      <xs:documentation>An entry in this DataElement's
documentation.</xs:documentation>
                                                </xs:annotation>
                                                <xs:complexType>

      <xs:simpleContent>
                                                      <xs:extension
base="xs:string">
                                                      <xs:attribute
name="locale" type="de:Locales"/>
                                                      <xs:attribute
name="type" type="de:DocumentationTypes"/>
                                                      <xs:attribute
name="mimeType" type="de:DocumentationMimeTypes" use="optional"
default="text/plain"/>
                                                      </xs:extension>

      </xs:simpleContent>
                                                </xs:complexType>
                                          </xs:element>
                                    </xs:sequence>
                              </xs:complexType>
                        </xs:element>
                        <xs:element name="Representations">
                              <xs:annotation>
```

```
                                    <xs:documentation>List of
representations of this DataElement.  A representation is essentially a
schema in any format imaginable, although it is suggested that an XML
schema format is used.</xs:documentation>
                              </xs:annotation>
                              <xs:complexType>
                                    <xs:sequence>
                                          <xs:element
name="Representation" maxOccurs="unbounded">
                                                <xs:annotation>

      <xs:documentation>A structure Representation of this DataElement.
It is recomended that the type attribute of this element be the
namespace value for the schema language being used.  For xsd, that
would be http://www.w3.org/2001/XMLSchema.</xs:documentation>
                                                </xs:annotation>
                                                <xs:complexType>

      <xs:simpleContent>
                                                      <xs:extension
base="xs:string">
                                                      <xs:attribute
name="type" type="xs:string"/>
                                                      <xs:attribute
name="context" type="xs:string"/>
                                                      </xs:extension>

      </xs:simpleContent>
                                                </xs:complexType>
                                          </xs:element>
                                    </xs:sequence>
                              </xs:complexType>
                        </xs:element>
                  </xs:sequence>
                  <xs:attribute name="home" type="xs:anyURI"
use="required"/>
                  <xs:attribute name="id" type="xs:string"
use="required"/>
            </xs:complexType>
      </xs:element>
      <xs:simpleType name="IdentifierTypes">
            <xs:annotation>
                  <xs:documentation>List of acceptable Identifier
types.</xs:documentation>
            </xs:annotation>
            <xs:restriction base="xs:string">
                  <xs:enumeration value="responsibleOrgURL"/>
                  <xs:enumeration value="submittingOrgURL"/>
                  <xs:enumeration value="ElementIdentifier"/>
                  <xs:enumeration value="DataDictionaryName"/>
                  <xs:enumeration value="EntityName"/>
            </xs:restriction>
      </xs:simpleType>
      <xs:simpleType name="PropertyNames">
            <xs:annotation>
```

```
                  <xs:documentation>List of property names in use.
This can also be extended. The following is an example only of how to
restrict the list to known types.</xs:documentation>
            </xs:annotation>
            <xs:restriction base="xs:string">
                  <xs:enumeration value="version.major"/>
                  <xs:enumeration value="version.minor"/>
                  <xs:enumeration value="version.incremental"/>
                  <xs:enumeration value="registration.status"/>
                  <xs:enumeration value="domain"/>
                  <xs:enumeration value="topic"/>
                  <xs:enumeration value="familiar.name"/>
                  <xs:enumeration value="synonyms"/>
            </xs:restriction>
      </xs:simpleType>
      <xs:simpleType name="Locales">
            <xs:annotation>
                  <xs:documentation>List of acceptable locales.  Used
primarily for choosing an appropriate culture for Documentation Entries.
This should point at an actual code list of values (perhaps stored in a
registry) and use xs:import.  The following is for example
only.</xs:documentation>
            </xs:annotation>
            <xs:restriction base="xs:string">
                  <xs:enumeration value="en_CA"/>
                  <xs:enumeration value="en"/>
                  <xs:enumeration value="en_GB"/>
                  <xs:enumeration value="en_US"/>
                  <xs:enumeration value="fr_FR"/>
                  <xs:enumeration value="fr"/>
                  <xs:enumeration value="fr_CA"/>
            </xs:restriction>
      </xs:simpleType>
      <xs:simpleType name="DocumentationMimeTypes">
            <xs:annotation>
                  <xs:documentation>List of recognized documentation
formats.  Remember to use CDATA when doing anything other than
text/plain.</xs:documentation>
            </xs:annotation>
            <xs:restriction base="xs:string">
                  <xs:enumeration value="text/plain"/>
                  <xs:enumeration value="text/html"/>
                  <xs:enumeration value="text/xml"/>
                  <xs:enumeration value="application/pdf"/>
                  <xs:enumeration value="application/ms-word"/>
                  <xs:enumeration value="text/rtf"/>
                  <xs:enumeration value="application/octet-stream"/>
            </xs:restriction>
      </xs:simpleType>
      <xs:simpleType name="DocumentationTypes">
            <xs:annotation>
                  <xs:documentation>List of documentation
types.</xs:documentation>
            </xs:annotation>
            <xs:restriction base="xs:string">
                  <xs:enumeration value="comment"/>
                  <xs:enumeration value="note"/>
```

```
                  <xs:enumeration value="instruction"/>
                  <xs:enumeration value="other"/>
                  <xs:enumeration value="warning"/>
                  <xs:enumeration value="copyright"/>
                  <xs:enumeration value="restrictions"/>
                  <xs:enumeration value="description"/>
                  <xs:enumeration value="abstract"/>
            </xs:restriction>
      </xs:simpleType>
</xs:schema>
```

Appendix "B" – Sample Java Code for Extracting BIE's from Data Elements.

```
package com.adobe.assembly;

import org.jdom.Element;
import org.jdom.Namespace;
import org.jdom.JDOMException;
import org.jdom.input.SAXBuilder;

import java.io.InputStream;
import java.io.IOException;
import java.util.*;

/**
 * Object representation of an XML DataElement.
 *
 * @author Matthew MacKenzie
 */
public class DataElement {
    public static final String DATA_ELEMENT_NAMESPACE =
"http://ns.cpsin.org/data-element/1.0";
    public static final String XSD_NAMESPACE =
"http://www.w3.org/2001/XMLSchema";
    private InputStream xmlStream;
    private Element rootNode;
    private List representations;
    private static final Namespace DATA_ELEMENT_NS =
Namespace.getNamespace(DataElement.DATA_ELEMENT_NAMESPACE);
    private static final Namespace XSD_NS =
Namespace.getNamespace(DataElement.XSD_NAMESPACE);

    private static final String REPRESENTATIONS_TAG = "Representations";
    private static final String REPRESENTATION_TAG = "Representation";
    private static final String SCHEMA_TAG = "schema";

    public DataElement(InputStream xmlStream) {
        this.xmlStream = xmlStream;
    }

    /**
     * Retrieves a representation give type and context.  If the
     * type matches XSD_NAMESPACE, the bare XSD is returned, otherwise
     * the whole Representation element is returned.  If nothing exists for
     * the given parameters, null is returned.
     */
    public Element retrieveRepresentation(String type, String context) throws
JDOMException, IOException {
        if (this.rootNode == null) {
            if (this.xmlStream == null)
                throw new IOException("XML Stream is null!");

            this.rootNode = new
SAXBuilder().build(this.xmlStream).getRootElement();

            if
(!this.rootNode.getNamespace().equals(DataElement.DATA_ELEMENT_NS))
                throw new JDOMException("Root node is not in the right
namespace (" + DataElement.DATA_ELEMENT_NAMESPACE + ")");
```

```
        }

        if (this.representations == null) {
            Element repsXml =
this.rootNode.getChild(DataElement.REPRESENTATIONS_TAG,
DataElement.DATA_ELEMENT_NS);

            this.representations =
repsXml.getChildren(DataElement.REPRESENTATION_TAG,
DataElement.DATA_ELEMENT_NS);
            //FOR DEBUG ONLY
            //System.out.println("Found " + this.representations.size()
            //                   + " representation(s) of this data element."
            //                   + "  The one below is for context: "
            //                   + context);

            if (this.representations == null)
                return null;
        }

        Iterator repIter = this.representations.iterator();

        while (repIter.hasNext()) {
            Element rep = (Element) repIter.next();
            if (rep.getAttributeValue("type").equals(type) &&
rep.getAttributeValue("context").equals(context)) {
                if (type.equals(DataElement.XSD_NAMESPACE)) {
                    return (Element)rep.getChild("schema",
DataElement.XSD_NS).detach();
                }
                return (Element)rep.detach();
            }
        }
        return null;

    }

    /**
     * Returns a merged XML schema from an array of schema fragments.
     *
     * @param representations Schema fragments.
     * @param rootElementName Name of root element.
     * @return an XML schema.  NOTE: valid XML returned, not necessarily valid
XSD!
     */
    public static Element getXMLSchema(Element[] representations, String
rootElementName) {
        Element schema = new Element(DataElement.SCHEMA_TAG, XSD_NS);
        Element rootElement = new Element("element", XSD_NS);
        rootElement.setAttribute("name", rootElementName);
        for (int i = 0; i < representations.length; i++) {
            Element schemaEl = (Element)representations[i].detach();
            List addEls = new ArrayList();
            if (schemaEl.getName().equals("schema")) {
                List schemaComp = schemaEl.getChildren();
                for (Iterator iterator = schemaComp.iterator();
iterator.hasNext();) {
                    Element e =  (Element)iterator.next();
                    addEls.add(((Element)e.clone()).detach());

                }
            }
```

```
            else addEls.add(schemaEl);
            rootElement.addContent(addEls);
        }
        schema.addContent(rootElement);
        return schema;
    }

}
```