UN/CEFACT – TMG
UML Profile for Core Components (BCSS)
Candidate for Version 1.0
Working Draft for public review
2006-10-31

Draft - Not For Implementation
Table of Contents

1 This Document ...................................................................................................................... 4
  1.1 Context .......................................................................................................................... 4
  1.2 Conventions .................................................................................................................. 4

2 Project Team Participants ................................................................................................... 5
  2.1 Disclaimer ..................................................................................................................... 5
  2.2 Contact Information ...................................................................................................... 5

3 Introduction ...................................................................................................................... 6
  3.1 Summary of Contents of Document .............................................................................. 6
  3.2 Audience ...................................................................................................................... 6
  3.3 Related Documents ...................................................................................................... 6

4 Objectives ....................................................................................................................... 7
  4.1 Goals ............................................................................................................................ 7
  4.2 Requirements ............................................................................................................... 7
  4.3 Caveats and Assumptions ............................................................................................ 7

5 Overview .......................................................................................................................... 8
  5.1 The model interchange problem ................................................................................... 8
  5.2 The UML profile approach .......................................................................................... 8

6 Core Component Module ................................................................................................. 10
  6.1 Conceptual Metamodel ............................................................................................... 10
  6.2 Relationships and dependencies ................................................................................... 10
  6.3 Partitioning ................................................................................................................... 10
  6.4 Common Package – UML Profile ............................................................................... 11
      6.4.1 Stereotypes and Tag Definitions (normative) ..................................................... 11
      6.4.2 Constraints (normative) ..................................................................................... 14
      6.4.3 Common Package – Mapping to CCTS ............................................................ 16
  6.5 Data Types Package – UML Profile ............................................................................ 18
      6.5.1 Stereotypes and Tag Definitions (normative) ..................................................... 18
      6.5.2 Constraints (normative) ..................................................................................... 21
      6.5.3 Data Type Package - Mapping to CCTS ............................................................ 22
  6.6 Management Package – UML Profile ......................................................................... 27
      6.6.1 Stereotypes and Tag Definitions (normative) ..................................................... 27
      6.6.2 Constraints (normative) ..................................................................................... 30
      6.6.3 Packages & Namespace Conventions for libraries ............................................. 32
  6.7 OCL methods used in the BCSS Profile ...................................................................... 33
  6.8 Context Package – UML Profile ................................................................................. 38
  6.9 Relationship to UMM ................................................................................................... 38
1 This Document

1.1 Context

The original goal of the BCSS project was to provide a “Business Collaboration Specification Schema” as a formalized solution to validate collaboration- and information models against the metamodels they are based on.

UN/CEFACT metamodels for business information (CCTS) and business process modelling (UMM) provide conceptual metamodels. For these to be usable with common modelling tools that support UML and to be able to use UML infrastructure for validation the conceptual models need to be mapped to the UML meta-model. This document provides a mapping of the CCTS to UML as a formal UML profile.

1.2 Conventions

The keywords MUST, MUST NOT, REQUIRED, SHALL, SHALL NOT, SHOULD, SHOULD NOT, RECOMMENDED, MAY and OPTIONAL, when they appear in this document, are to be interpreted as described in [RFC2119] as quoted here:

- MUST: This word, or the terms "REQUIRED" or "SHALL", means that the definition is an absolute requirement of the specification.
- MUST NOT: This phrase, or the phrase "SHALL NOT", means that the definition is an absolute prohibition of the specification.
- SHOULD: This word, or the adjective "RECOMMENDED", means that there may exist valid reasons in particular circumstances to ignore a particular item, but the full implications MUST be understood and carefully weighed before choosing a different course.
- SHOULD NOT: This phrase, or the phrase "NOT RECOMMENDED", means that there may exist valid reasons in particular circumstances when the particular behavior is acceptable or even useful.
but the full implications should be understood and the case carefully weighed before implementing any behavior described with this label.

- MAY: This word, or the adjective "OPTIONAL", means that an item is truly optional. One vendor may choose to include the item because a particular marketplace requires it or because the vendor feels that it enhances the product while another vendor may omit the same item. An implementation that does not include a particular option MUST be prepared to interoperate with another implementation which does include the option, though perhaps with reduced functionality. In the same vein an implementation that does include a particular option MUST be prepared to interoperate with another implementation which does not include the option (except, of course, for the feature the option provides).

2 Project Team Participants

2.1 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 specification.

2.2 Contact Information

Editor       Steve Capell       steve.capell@redwahoo.com
Contributors Mike Conroy        michael.conroy@wanadoo.fr
Jens Dietrich jens.dietrich@gmx.de
Philipp Liegl philipp.liegl@gmx.at
Harry Moyer   harry.moyer@redwahoo.com
Joern Guy Süß joern.guy.suess@web.de
Anders Tell   andert@toolsmiths.se
Jim Wilson    jim.wilson@cidx.org
3 Introduction

3.1 Summary of Contents of Document

3.2 Audience

The target audience for this document includes members of the UN/CEFACT TMG, other UN/CEFACT working groups, the UN/CEFACT Forum management group, and the wider community of modeling tool vendors.

3.3 Related Documents

UN/CEFACT Specifications

UMM Base Module 1.0

UMM Foundation Module 1.0

CCTS 2.01

ATG2 XML NDR

OMG Specifications

UML 2.0
http://www.omg.org/cgi-bin/doc?formal/05-07-04
http://www.omg.org/cgi-bin/doc?formal/05-07-05

UML 1.4.2 (ISO)
http://www.omg.org/cgi-bin/doc?formal/04-07-02

MOF 1.3
http://www.omg.org/cgi-bin/doc?formal/00-04-03

XMI 1.2

Other Related Documents

MDR
http://mdr.netbeans.org/

JMI
http://jcp.org/aboutJava/communityprocess/final/jsr040/index.html

Eclipse UML2 Project
http://www.eclipse.org/uml2/
4 Objectives

4.1 Goals

The business goals of this specification are:

• To make CCTS compliant information modelling accessible to a broad user base through standard UML tool support.
• To support easy interchange of information models between different UML tools.
• To support validation of the structure and semantics of information models against the CCTS.

These goals are achieved through the development of a formal UML profile for CCTS that includes stereotypes, tagged values and OCL constraints.

4.2 Requirements

This specification is guided by the following key requirements:

• That accessibility to a large community through availability of high quality tools is the key objective.
• That the UML profile should minimise complexity for the business user.
• That the UML profile provides a complete representation of data required by CCTS (no information loss).
• The UML profile for core components complements the base and foundation profiles for UMM.
• The UML profile may need to define different levels of compliance – depending on the requirements of the modeler and the capabilities of the tool.
• The UML profile shall be implementable by the widest group of tool vendors.
• The UML profile will support the generation of XML schema from the model that are compliant with ATG2 XML Naming and Design Rules.

4.3 Caveats and Assumptions

This specification makes the following assumptions:

• That the XMI generated by modeling tools will be compliant with a reference XMI implementation (either based on JSR40 (JMI), the Eclipse UML2 project or the UML 2.0 Tools Certification program). The feasibility of this approach will be evaluated as part of the implementation verification phase of this project.
• That most modeling tools do not evaluate OCL constraints against model data. Accordingly, validation of CCTS semantics as defined by the OCL constraints in this specification will normally only be possible using either an external validation service or a custom plug-in.
• The UML profile does not specify requirements for diagram interchange.
5 Overview

5.1 The model interchange problem

The OMG XML Metadata Interchange (XMI) specification has been defined to support exchange of model data between different tools. However:

- There are several versions of XMI and different tools may support different versions.
- The XMI is just a description of a model instance in terms of the reference UML meta-model. But since there are several versions of the UML reference model, files exported from one tool may not be understood by another.
- We wish to use UML as a notation to describe information and process models. We want to describe things like “BCC”, “ABIE”, “Transactions”, “roles”, etc. But these are not UML elements, they are CCTS or UMM concepts. So if one modeler chooses to represent a BBIE in one way and a different modeler chooses to represent the same BIE using a different UML model element then models will not be meaningful or re-usable amongst different users. The goal of generating deployment schema like ATG2 XSD, or BPSS from models cannot be realized.

So, to have useful interchange at the Model level we need:

- To specify supported UML versions that will be used. This specification supports UML 1.4.2 (which is also the ISO standard version) and UML2.
- More importantly, to define precise mappings from CCTS / UMM concepts to corresponding UML elements and to provide these as a “UML profile” that consists of stereotypes, tagged values and OCL constraints. That is the purpose of this specification.
- And finally to propose a XMI implementation as a reference that ensures consistent model exchange.

So what exactly will this specification deliver?

- A normative specification (this document) that defines the UML profile for CCTS as a set of model diagrams showing stereotypes and tagged values plus a set of OCL constraints that describe the semantic restrictions imposed by CCTS.
- An XMI file containing the UML profiles that can be imported into any compliant modelling tool (as part of the final specification).
- A set of compliance levels with corresponding test cases that can be used by implementers to test their products.

5.2 The UML profile approach

The approach taken by this specification to deliver on the goal of interchange of structurally and semantically valid information models is shown in the diagram below.
The CCTS v2.0.1 is the reference for the structural and semantic constraints that the UML profile should reflect.

The UML profile is provided as an XMI Template file containing stereotypes, tagged values, & constraints. The UML profile supports UML 1.4.2 and UML2.

Modellers will use the template together with library components (that also conform to this profile) to build their CCTS compliant information models.

An interoperable XMI representation of the model can be generated from any tool that supports JMI XMI rules. The XMI can be sent to a validation service, submitted to a repository, or exchanged with other modellers.

ATG2 XML NDR compliant deployment schema can be generated directly from the model via JMI.

Figure 2 – UML Profile Usage
6 Core Component Module

This section is normative. This section provides the detailed definition of the UML profile for Core Components.

6.1 Conceptual Metamodel

The conceptual meta-model to which this profile must comply is defined in the Core Component Technical Specification Version 2.1 and is not repeated here.

6.2 Relationships and dependencies

The CCTS module uses the RegistryObject and BusinessLibrary elements from the base module.

6.3 Partitioning
6.4 Common Package – UML Profile

6.4.1 Stereotypes and Tag Definitions (normative)

The following diagram shows a visualization of the Common Package to provide an overview of the defined elements. Note that the diagram shows only the structure of the UML Profile. Additional constraints to define compliance with the semantics of the CCTS meta-model are provided in a subsequent section.

The sections below provide details on each stereotype in the Common Package. Both abstract and concrete stereotypes are described. However OCL constraints are only defined for concrete elements because these are the only ones that will exist in an actual instance model.

6.4.1.1 CC

<table>
<thead>
<tr>
<th>Stereotype</th>
<th>CC (Abstract)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Base Class</td>
<td>ModelElement  (from UML 1.4.2 / 2.0 meta-model)</td>
</tr>
<tr>
<td>Parent</td>
<td>Management::RegistryObject (from UMM base module)</td>
</tr>
<tr>
<td>Description</td>
<td>The CC (Core Component) element is a superclass to provide attributes common to BCC, ASCC, ACC, and CDT types.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Tag Definitions</th>
<th>Tag Definition</th>
<th>Type</th>
<th>Multiplicity</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>DictionaryEntryName</td>
<td>string</td>
<td>1:1</td>
<td>Official Name of the Core Component.</td>
</tr>
</tbody>
</table>

6.4.1.2 BIE

<table>
<thead>
<tr>
<th>Stereotype</th>
<th>BIE (Abstract)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Base Class</td>
<td>ModelElement  (from UML 1.4.2 / 2.0 meta-model)</td>
</tr>
</tbody>
</table>
Parent: Management::RegistryObject (from UMM base module)

**Description**
The BIE (Business Information Entity) element is a superclass to provide attributes common to BBIE, ASBIE, ABIE and QDT types.

<table>
<thead>
<tr>
<th>Tag Definitions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tag Definition</td>
</tr>
<tr>
<td>DictionaryEntryName</td>
</tr>
</tbody>
</table>

### 6.4.1.3 BCC

**Stereotype**: BCC

**Base Class**: Attribute (from UML 1.4.2 / 2.0 meta-model)

**Parent**: CC

**Description**
The BCC (Basic Core Component) is a singular business characteristic of an ACC (Aggregate Core Component).

<table>
<thead>
<tr>
<th>Tag Definitions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tag Definition</td>
</tr>
<tr>
<td>N/A</td>
</tr>
</tbody>
</table>

### 6.4.1.4 ACC

**Stereotype**: ACC

**Base Class**: Class (from UML 1.4.2 / 2.0 meta-model)

**Parent**: CC

**Description**
The ACC (Aggregate Core Component) is a collection of related business information that together carry a distinct meaning.

<table>
<thead>
<tr>
<th>Tag Definitions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tag Definition</td>
</tr>
<tr>
<td>N/A</td>
</tr>
</tbody>
</table>

### 6.4.1.5 ASCC

**Stereotype**: ASCC

**Base Class**: Association (from UML 1.4.2 / 2.0 meta-model)

**Parent**: CC

**Description**
The ASCC (Association Core Component) represents a complex characteristic of an associated ACC. The ACC describes the structure of the ASCC.

<table>
<thead>
<tr>
<th>Tag Definitions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tag Definition</td>
</tr>
<tr>
<td>N/A</td>
</tr>
</tbody>
</table>

### 6.4.1.6 BBIE

**Stereotype**: BBIE

**Base Class**: Attribute (from UML 1.4.2 / 2.0 meta-model)
### 6.4.1.7 ABIE

<table>
<thead>
<tr>
<th>Stereotype</th>
<th>ABIE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Base Class</td>
<td>Class (from UML 1.4.2 / 2.0 meta-model)</td>
</tr>
<tr>
<td>Parent</td>
<td>BIE</td>
</tr>
<tr>
<td>Description</td>
<td>The ABIE (Aggregate Business Information Entity) is a collection of related business information that together express a business meaning in a specific context. An ABIE is related to the ACC from which it is derived.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Tag Definitions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tag Definition</td>
</tr>
<tr>
<td>N/A</td>
</tr>
</tbody>
</table>

### 6.4.1.8 ASBIE

<table>
<thead>
<tr>
<th>Stereotype</th>
<th>ASBIE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Base Class</td>
<td>Association (from UML 1.4.2 / 2.0 meta-model)</td>
</tr>
<tr>
<td>Parent</td>
<td>BIE</td>
</tr>
<tr>
<td>Description</td>
<td>The ASBIE (Association Business Information Entity) represents a complex business characteristic of an associated ABIE that describes it’s structure. An ASBIE is related to the ASCC from which it is derived.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Tag Definitions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tag Definition</td>
</tr>
<tr>
<td>N/A</td>
</tr>
</tbody>
</table>

### 6.4.1.9 basedOn

<table>
<thead>
<tr>
<th>Stereotype</th>
<th>basedOn</th>
</tr>
</thead>
<tbody>
<tr>
<td>Base Class</td>
<td>Dependency (from UML 1.4.2 / 2.0 meta-model)</td>
</tr>
<tr>
<td>Parent</td>
<td>N/A</td>
</tr>
</tbody>
</table>
| Description | The basedOn stereotype is used to define the relationships between
- A QDT and the CDT upon which it is based
- An ABIE and the ACC upon which it is based |

<table>
<thead>
<tr>
<th>Tag Definitions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tag Definition</td>
</tr>
<tr>
<td>N/A</td>
</tr>
</tbody>
</table>
6.4.2 Constraints (normative)

A BCC type must be typed with a class of stereotype <<CDT>>.

**UML 1.4**

context Attribute

```plaintext
inv BCC_CDT:
    self.isBCC() implies
    self.type.oclAsType(ModelElement).isCDT()
```

**UML 2.0**

context Property

```plaintext
inv BCC_CDT:
    self.isBCC() implies
    self.type.oclAsType(Element).isCDT()
```

An ACC must contain one or more attributes of stereotype <<BCC>>. There must be no other attributes.

**UML 1.4**

context Classifier

```plaintext
inv ACC_OneOrMoreBCC:
    self.isACC() implies
    self.typedFeature->select(attribute | attribute.isBCC())->size() >= 1
    and
    self.typedFeature->select(attribute | not attribute.isBCC())->size() = 0
```

**UML 2.0**

context Class

```plaintext
inv ACC_OneOrMoreBCC:
    self.isACC() implies
    self.ownedAttribute->select(property | property.isBCC())->size() >= 1
    and
    self.ownedAttribute->select(property | not property.isBCC())->size() = 0
```

A BBIE type must be typed with a class of stereotype either <<QDT>> or <<CDT>>.

**UML 1.4**

context Attribute

```plaintext
inv BBIE_QDT:
    self.isBBIE() implies
    self.type.oclAsType(ModelElement).isCDT() or
    self.type.oclAsType(ModelElement).isQDT()
```

**UML 2.0**

context Property

```plaintext
inv BBIE_QDT:
    self.isBBIE() implies
    self.type.oclAsType(Element).isCDT() or
    self.type.oclAsType(Element).isQDT()
```
An ABIE must contain one or more attributes of stereotype <<BBIE>>. There must be no other attributes.

### UML 1.4
context Classifier
inv ABIE_OneOrMoreBBIE:
  self.isABIE() implies
  self.typedFeature->select(attribute | attribute.isBBIE())->size() >= 1
  and
  self.typedFeature->select(attribute | not attribute.isBBIE())->size() = 0

### UML 2.0
context Class
inv ABIE_OneOrMoreBBIE:
  self.isABIE() implies
  self.ownedAttribute->select(property | property.isBBIE())->size() >= 1
  and
  self.ownedAttribute->select(property | not property.isBBIE())->size() = 0

An ABIE must have a dependency of stereotype <<basedOn>> with an ACC as the target.

### UML 1.4
context Class
inv ABIE_basedOnACC:
  self.isABIE() implies
  self.supplierDependency->one(isBasedOnToACC())

### UML 2.0
context NamedElement
inv ABIE_basedOnACC:
  self.isABIE() implies
  self.supplierDependency->one(isBasedOnToACC())

The originating association end of a stereotyp ASCC must be set to AggregationKind::composite.

### UML 1.4
context Association
inv ASCCAssociationEnd_composite:
  self.isASCC() implies
  self.isComposition()
Two associations emanating from the same ACC/ABIE and leading to the same ACC/ABIE must not have the same role name. Two associations which emanate from the same ACC/ABIE and lead to the same ACC/ABIE and have no role names are not valid.

### UML 1.4

**context** Class

**inv** noTwoAssociationsWithSameRoleNameLeadingToACC:

```plaintext
self.isACC() or self.isABIE() implies
self.association->forAll(a1, a2 | a1 <> a2 implies a1.name <> a2.name and a1.participant <> a2.participant)
```

### UML 2.0

**context** Class

**inv** noTwoAssociationsWithSameRoleNameLeadingToACC:

```plaintext
self.isACC() or self.isABIE() implies
self.association->forAll(a1, a2 | a1 <> a2 implies a1.name <> a2.name and a1.memberEnd <> a2.memberEnd)
```

## 6.4.3 Common Package – Mapping to CCTS

<table>
<thead>
<tr>
<th>CCTS Construct</th>
<th>UML 1.4.2 Implementation</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Parent objects - properties inherited by CDT, QDT, ACC, BCC, ASCC, ABIE, BBIE, ASBIE</strong></td>
<td></td>
</tr>
<tr>
<td>Registry Class - Definition</td>
<td>Implemented using the UML 1.4.2 “Comment” element.</td>
</tr>
<tr>
<td></td>
<td>UML modelling tools normally implement the “Comment” element as a notes area on data entry window for all model elements. Accordingly business users may find it simpler to use the notes field for CCTS definitions.</td>
</tr>
<tr>
<td>Registry Class – Dictionary EntryName</td>
<td>Tagged value with name = “dictionaryEntryName”.</td>
</tr>
<tr>
<td></td>
<td>Implemented at CC/BIE level because registry object is re-used by UMM Foundation. Note that no extra information is introduced by this tagged value because the dictionary entry name is already derivable from class, attribute, and data type names. Business users may prefer to leave this value out and then apply a transform to the resulting XMI.</td>
</tr>
</tbody>
</table>

### Core Component Elements

<table>
<thead>
<tr>
<th>Aggregate Core Component – Object Class Term</th>
<th>Name of a UML class with stereotype &lt;&lt;ACC&gt;&gt;. Note that this is just the object class term and should not include the “.Details” suffix.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Basic Core Component – property term</td>
<td>Name of an UML attribute with stereotype &lt;&lt;BCC&gt;&gt;. Note that this is just the property term and should not include the object class</td>
</tr>
<tr>
<td><strong>Special Note:</strong> in the rare case where an ACC has two BCCs with the same property term then the representation term should be added to the UML attribute name in order to ensure uniqueness of UML attributes in a class.</td>
<td></td>
</tr>
<tr>
<td>-------------------</td>
<td>-----------------------------------</td>
</tr>
<tr>
<td><strong>Basic Core Component - cardinality</strong></td>
<td>UML cardinality of an attribute with stereotype &lt;&lt;BCC&gt;&gt;.</td>
</tr>
</tbody>
</table>
| **Basic Core Component – representation term** | UML type of an attribute with stereotype <<BCC>>.  
The type of the BCC must be another class of stereotype <<CDT>> (Core Data Type). Note that the representation term is equal to the name of the CDT class. |
| **Association Core Component – property term** | UML aggregation source role name of an aggregation of stereotype <<ASCC>>.  
Note that the name is just the property term and not the dictionary entry name. |

**Business Information Entity Elements**

| **ABIE / BBIE / ASBIE / QDT Qualifier Terms** | The qualifier prefix (characters before the underscore) of the ABIE, BBIE, ASBIE and QDT name. |
| **Aggregate Business Information Entity – Object class term** | Name of a UML class with stereotype <<ABIE>> (without qualifier prefix if it exists)  
Note: The ABIE class name is the qualified object class term - shall be equal to the object class name of the ACC upon which this ABIE is based with an optional prefix equal to the qualifier term. |
| **Basic Business Information Entity – property term** | Name of a UML attribute with stereotype <<BBIE>> (without qualifier prefix if it exists)  
Note: The BBIE attribute name is the qualified property term - shall be equal to the property name of the BCC upon which this BBIE is based with an optional prefix equal to the qualifier term.  
Special Note: in the rare case where an ABIE has two BBIEs with the same property term then the representation term should be added to the UML attribute name in order to ensure uniqueness of UML attributes in a class. |
| **Basic Business Information Entity – representation term** | UML type of an attribute with stereotype <<BBIE>>.  
The type of the BBIE must be another class of stereotype <<CDT>> (Core Data Type) or <<QDT>> (Qualified Data Type).  
Note that the representation term is equal to the name of the CDT class (or
<table>
<thead>
<tr>
<th>QDT class without qualifier term)</th>
<th>Association Business Information Entity – property term</th>
</tr>
</thead>
<tbody>
<tr>
<td>UML aggregation source role name of an aggregation of stereotype &lt;&lt;ASBIE&gt;&gt;. Note that the name is just the property term and not the dictionary entry name.</td>
<td></td>
</tr>
<tr>
<td>UML Dependency of stereotype &lt;&lt;basedOn&gt;&gt; between ABIE and ACC with target ACC.</td>
<td></td>
</tr>
<tr>
<td>“basis” relationship between ABIE and ACC</td>
<td></td>
</tr>
<tr>
<td>No explicit association. BBIE property terms are the same as BCC property terms and are unique within the aggregate. Therefore the explicit dependency between ABIE and ACC is sufficient to infer this relationship.</td>
<td></td>
</tr>
<tr>
<td>Practical note: in most modeling tools a dependency between attributes is not supported. Workarounds are generally inconvenient to the modeler.</td>
<td></td>
</tr>
<tr>
<td>“basis” relationship between BBIE and BCC</td>
<td></td>
</tr>
<tr>
<td>No explicit association. ASBIE property terms are the same as ASCC property terms and are unique within the aggregate. Therefore the explicit dependency between ABIE and ACC is sufficient to infer this relationship</td>
<td></td>
</tr>
</tbody>
</table>

### 6.5 Data Types Package – UML Profile

#### 6.5.1 Stereotypes and Tag Definitions (normative)

The following diagram shows a visualization of the Data Types Package to provide an overview of the defined elements. Note that the diagram shows only the structure of the UML Profile. Additional constraints to define compliance with the semantics of the CCTS meta-model are provided in a subsequent section.

![Data Types Package Diagram](image)

The sections below provide details on each stereotype in the Data Types Package. Both abstract and concrete stereotypes are described. However OCL constraints are only defined for concrete elements because these are the only ones that will exist in an actual instance model.
### 6.5.1.1 CDT

<table>
<thead>
<tr>
<th>Stereotype</th>
<th>CDT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Base Class</td>
<td>Class (from UML 1.4.2 / 2.0 meta-model)</td>
</tr>
<tr>
<td>Parent</td>
<td>Common::CC</td>
</tr>
<tr>
<td>Description</td>
<td>The CDT (Core Data Type) element is a complex data type that must be one of the approved Core Component Types listed in the CCTS v 2.0.1 section 8.</td>
</tr>
<tr>
<td>Tag Definitions</td>
<td></td>
</tr>
<tr>
<td>Tag Definition</td>
<td>Type</td>
</tr>
<tr>
<td>N/A</td>
<td></td>
</tr>
</tbody>
</table>

### 6.5.1.2 QDT

<table>
<thead>
<tr>
<th>Stereotype</th>
<th>QDT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Base Class</td>
<td>Class (from UML 1.4.2 / 2.0 meta-model)</td>
</tr>
<tr>
<td>Parent</td>
<td>Common::BIE</td>
</tr>
<tr>
<td>Description</td>
<td>The QDT (Qualified Data Type) element is a complex data type. It is a restriction of a corresponding CDT (Core Data Type) element.</td>
</tr>
<tr>
<td>Tag Definitions</td>
<td></td>
</tr>
<tr>
<td>Tag Definition</td>
<td>Type</td>
</tr>
<tr>
<td>N/A</td>
<td></td>
</tr>
</tbody>
</table>

### 6.5.1.3 CON

<table>
<thead>
<tr>
<th>Stereotype</th>
<th>CON</th>
</tr>
</thead>
<tbody>
<tr>
<td>Base Class</td>
<td>Attribute (from UML 1.4.2 / 2.0 meta-model)</td>
</tr>
<tr>
<td>Parent</td>
<td>N/A</td>
</tr>
<tr>
<td>Description</td>
<td>The CON (Content Component) element carries the actual content. The type of the CON element shall be a class of stereotype &lt;&lt;PRIM&gt;&gt;.</td>
</tr>
<tr>
<td>Tag Definitions</td>
<td></td>
</tr>
<tr>
<td>Tag Definition</td>
<td>Type</td>
</tr>
<tr>
<td>N/A</td>
<td></td>
</tr>
</tbody>
</table>

---

Page 19 of 46
### 6.5.1.4 SUP

<table>
<thead>
<tr>
<th>Stereotype</th>
<th>SUP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Base Class</td>
<td>Attribute (from UML 1.4.2 / 2.0 meta-model)</td>
</tr>
<tr>
<td>Parent</td>
<td>N/A</td>
</tr>
<tr>
<td>Description</td>
<td>The SUP (Supplementary Component) element provides additional meaning to the content component of a CDT or QDT.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Tag Definitions</th>
<th>Tag Definition</th>
<th>Type</th>
<th>Multiplicity</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>N/A</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### 6.5.1.5 PRIM

<table>
<thead>
<tr>
<th>Stereotype</th>
<th>PRIM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Base Class</td>
<td>Class (from UML 1.4.2 / 2.0 meta-model)</td>
</tr>
<tr>
<td>Parent</td>
<td>N/A</td>
</tr>
<tr>
<td>Description</td>
<td>The PRIM (Primitive Type) represents one of the CCTS defined primitive types. All CON and SUP types must be drawn from the collection of objects of stereotype PRIM.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Tag Definitions</th>
<th>Tag Definition</th>
<th>Type</th>
<th>Multiplicity</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>N/A</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### 6.5.1.6 ENUM

<table>
<thead>
<tr>
<th>Stereotype</th>
<th>ENUM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Base Class</td>
<td>Class (from UML 1.4.2 / 2.0 meta-model)</td>
</tr>
<tr>
<td>Parent</td>
<td>N/A</td>
</tr>
<tr>
<td>Description</td>
<td>The ENUM (Enumeration Type) represents an enumeration type. The ENUM stereotype is used to define a restriction on either a content- or supplementary component of a QDT.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Tag Definitions</th>
<th>Tag Definition</th>
<th>Type</th>
<th>Multiplicity</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>N/A</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
6.5.2 Constraints (normative)

A CDT or QDT must contain exactly one content component (an attribute of stereotype <<CON>>)

UML 1.4
context Classifier
inv CDT_OneCON:
    self.isCDT or self.isQDT() implies
    self.typedFeature->select(attribute | attribute.isCON())->size() = 1

UML 2.0
context Class
inv CDT_OneCON:
    self.isCDT or self.isQDT() implies
    self.ownedAttribute->select(property | property.isCON())->size() = 1

A CDT or QDT must not contain any attributes other than those of stereotype <<CON>> or <<SUP>>.

UML 1.4
context Classifier
inv CDT_onlyCONorSUP:
    self.isCDT or self.isQDT() implies
    self.typedFeature->select(attribute | not attribute.isCON() or attribute.isSUP())->size() = 0

UML 2.0
context Class
inv CDT_onlyCONorSUP:
    self.isCDT or self.isQDT() implies
    self.ownedAttribute->select(property | not property.isCON() or property.isSUP())->size() = 0

A QDT must have a dependency (with stereotype <<basedOn>>) with one CDT with target CDT.

UML 1.4
context Class
inv QDT_CDTdependency:
    self.isQDT() implies
    self.supplierDependency->one(isBasedOnToCDT())

UML 2.0
context NamedElement
inv QDT_CDTdependency:
    self.isQDT() implies
    self.supplierDependency->one(isBasedOnToCDT())
The type of CDT attributes must be a class of stereotype <<PRIM>>.

**UML 1.4**
context Classifier
inv CDT_onlyPRIM:
    self.isCDT() implies
    self.typedFeature->forAll(attribute | attribute.isPRIM())

**UML 2.0**
context Class
inv CDT_onlyPRIM:
    self.isCDT() implies
    self.ownedAttribute->forAll(property | property.isPRIM())

The type of QDT attributes must be a class of stereotype either <<PRIM>> or <<ENUM>>

**UML 1.4**
context Classifier
inv QDT_onlyPRIMorENUM:
    self.isQDT() implies
    self.typedFeature->forAll(attribute | attribute.isPRIM() or attribute.isENUM())

**UML 2.0**
context Class
inv QDT_onlyPRIMorENUM:
    self.isQDT() implies
    self.ownedAttribute->forAll(property | property.isPRIM() or property.isENUM())

### 6.5.3 Data Type Package - Mapping to CCTS

<table>
<thead>
<tr>
<th>CCTS Construct</th>
<th>UML 1.4.2 Implementation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Core Component Type – Representation Term</td>
<td>Name of a UML class with stereotype &lt;&lt;CDT&gt;&gt;.</td>
</tr>
<tr>
<td>Data Type – Representation Term</td>
<td>Name of a UML class with stereotype &lt;&lt;QDT&gt;&gt; (without qualifier prefix if it exists).</td>
</tr>
<tr>
<td></td>
<td>Note: The QDT class name is the qualified representation term - shall be equal to the class name of the CDT upon which this QDT is based with an optional prefix equal to the qualifier term.</td>
</tr>
<tr>
<td>Data Type – Qualifier</td>
<td>The qualifier prefix (characters before the underscore) of the QDT name.</td>
</tr>
<tr>
<td>Term</td>
<td>Description</td>
</tr>
<tr>
<td>------</td>
<td>-------------</td>
</tr>
<tr>
<td>Basis relationship between Date Type and Core Component Type</td>
<td>UML association of stereotype &lt;&lt;basedOn&gt;&gt;.</td>
</tr>
<tr>
<td>Content Component</td>
<td>UML attribute of stereotype &lt;&lt;CON&gt;&gt; of class &lt;&lt;CDT&gt;&gt; or &lt;&lt;QDT&gt;&gt;</td>
</tr>
<tr>
<td>Supplementary Component</td>
<td>UML attribute of stereotype &lt;&lt;SUP&gt;&gt; of class &lt;&lt;CDT&gt;&gt; or &lt;&lt;QDT&gt;&gt;</td>
</tr>
<tr>
<td>CCTS Primitive type</td>
<td>UML class of stereotype &lt;&lt;PRIM&gt;&gt;.</td>
</tr>
<tr>
<td>CCTS restriction</td>
<td>UML OCL constraint on class of stereotype &lt;&lt;QDT&gt;&gt;.</td>
</tr>
<tr>
<td>CCTS restriction of type “enumeration”</td>
<td>UML class of stereotype &lt;&lt;ENUM&gt;&gt;. Note that, so far as CCTS is concerned, “enumeration” is just one of the allowed primitives. In this profile, the enumerated primitive type is represented as a class of stereotype &lt;&lt;ENUM&gt;&gt; for convenience and readability purposes.</td>
</tr>
</tbody>
</table>

### 6.5.3.1 Content and Supplementary Component Restrictions – Global OCL definitions

This specification requires that restrictions on content and supplementary components be expressed as OCL constraints on the corresponding instance of the <<CON>> or <<SUP>> stereotypes. CCTS defines a set of standard restriction types. Some of those restrictions are more difficult to express in OCL than others (eg Fractional Digits). Therefore a general approach is taken to hide this complexity from the modeler by defining global OCL functions that implement the constraints. The modeler then just calls the function using the CCTS terminology.
### Global OCL definitions for Content and Supplementary Component restrictions

**UML 1.4**

```oclet
package Foundation::Data_Types

context String

--Returns true if the length of the primitive type String equals the passed value
let Length(Length : Integer) : Boolean =
  self.size()=Length

--Returns true if the length of the primitive type String is greater than or equals the passed value
let MinimumLength(MinimumLength : Integer) : Boolean =
  self.size()>=MinimumLength

--Returns true if the length of the primitive type String is smaller than or equals the passed value
let MaximumLength(MaximumLength : Integer) : Boolean =
  self.size()<=MaximumLength

--Returns true if the total number of digits of the primitive type Decimal equals the passed value
let TotalDigits(TotalDigits : Integer) : Boolean =
  self.size()-1=TotalDigits

context Real

--Returns true if the number of digits after the comma equals the passed value
let FractionalDigits(FractionalDigits : Integer) : Boolean =
  self.oclAsType(String).size()
  self.floor().oclAsType(String).size())-1=FractionalDigits

--Returns true if the primitive type is greater than or equals the passed value
let MinimumInclusive(MinimumInclusiveDate : Real) :Boolean =
  self >= MinimumInclusiveDate

--Returns true if the primitive type is smaller than or equals the passed value
let MaximumInclusive(MaximumInclusiveDate : Real) :Boolean =
  self <= MaximumInclusiveDate

--Returns true if the primitive type is bigger than the passed value
let MinimumExclusive(MinimumExclusiveDate : Real) :Boolean =
  self > MinimumExclusiveDate

--Returns true if the primitive type is smaller than the passed value
let MaximumExclusive(MaximumExclusiveDate : Real) :Boolean =
  self < MaximumExclusiveDate
```
context String
--Returns true if the primitive type is greater than or equals the passed value
def:
  let MinimumInclusiveDate(MinimumInclusiveDate : string) : Boolean =
  self >= MinimumInclusiveDate

--Returns true if the primitive type is smaller than or equals the passed value
def:
  let MaximumInclusiveDate(MaximumInclusiveDate : string) : Boolean =
  self <= MaximumInclusiveDate

--Returns true if the primitive type is greater than the passed value
def:
  let MinimumExclusiveDate(MinimumExclusiveDate : string) : Boolean =
  self > MinimumExclusiveDate

--Returns true if the primitive type is smaller than the passed value
def:
  let MaximumExclusiveDate(MaximumExclusiveDate : string) : Boolean =
  self < MaximumExclusiveDate
endpackage

UML 2.0
package Classes:Kernel

context String
--Returns true if the length of the primitive type String equals the passed value
def:
  let Length(Length : Integer) : Boolean =
  self.size() = Length

--Returns true if the length of the primitive type String is greater than or equals the passed value
def:
  let MinimumLength(MinimumLength : Integer) : Boolean =
  self.size() >= MinimumLength

--Returns true if the length of the primitive type String is smaller than or equals the passed value
def:
  let MaximumLength(MaximumLength : Integer) : Boolean =
  self.size() <= MaximumLength

--Returns true if the total number of digits of the primitive type Decimal equals the passed value
def:
  let TotalDigits(TotalDigits : Integer) : Boolean =
  self.size() - 1 = TotalDigits

--Returns true if the number of digits after the comma equals the passed value
def:
  let FractionalDigits(FractionalDigits : Integer) : Boolean =
  (self.oclAsType(String).size() -
  self.floor().oclAsType(String).size()) - 1 = FractionalDigits
--Returns true if the primitive type is greater than or
--equals the passed value
def:
    let MinimumInclusive(MinimumInclusiveDate : Real) :Boolean =
    self >= MinimumInclusiveDate

--Returns true if the primitive type is smaller than or
--equals the passed value
def:
    let MaximumInclusive(MaximumInclusiveDate : Real) :Boolean =
    self <= MaximumInclusiveDate

--Returns true if the primitive type is bigger than the passed value
def:
    let MinimumExclusive(MinimumExclusiveDate : Real) :Boolean =
    self > MinimumExclusiveDate

--Returns true if the primitive type is smaller than the passed value
def:
    let MaximumExclusive(MaximumExclusiveDate : Real) :Boolean =
    self < MaximumExclusiveDate

--Returns true if the primitive type is greater than or
--equals the passed value
class String
def:
    let MinimumInclusiveDate(MinimumInclusiveDate : string) :Boolean =
    self >= MinimumInclusiveDate

--Returns true if the primitive type is smaller than or equals
--the passed value
def:
    let MaximumInclusiveDate(MaximumInclusiveDate : string) :Boolean =
    self <= MinimumInclusiveDate

--Returns true if the primitive type is greater than the passed value
def:
    let MinimumExclusiveDate(MinimumExclusiveDate : string) :Boolean =
    self > MinimumExclusiveDate

--Returns true if the primitive type is smaller than the passed value
def:
    let MaximumExclusiveDate(MaximumExclusiveDate : string) :Boolean =
    self < MaximumExclusiveDate

class endpackage

6.5.3.2 Template CCTS restrictions as OCL constraints

The table below lists the expressions that will be used by a business modeller to express CCTS compliant
constraints on content and supplementary components.

<table>
<thead>
<tr>
<th>CCTS Restriction</th>
<th>Applies to Primitive Type</th>
<th>Example OCL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Expression</td>
<td>String</td>
<td>Any valid OCL expression</td>
</tr>
<tr>
<td>Parameter</td>
<td>Type</td>
<td>Constraint</td>
</tr>
<tr>
<td>-------------------</td>
<td>---------</td>
<td>-------------------------------------------------</td>
</tr>
<tr>
<td>Length</td>
<td>String</td>
<td>self.Content.Length(4)</td>
</tr>
<tr>
<td>Minimum Length</td>
<td>String</td>
<td>self.Content.MinimumLength(4)</td>
</tr>
<tr>
<td>Maximum Length</td>
<td>String</td>
<td>self.Content.MaximumLength(4)</td>
</tr>
<tr>
<td>Enumeration</td>
<td>String</td>
<td>Not implemented as OCL. Please use &lt;&lt;enum&gt;&gt; stereotype.</td>
</tr>
<tr>
<td>Total Digits</td>
<td>Decimal</td>
<td>self.content.TotalDigits(10)</td>
</tr>
<tr>
<td>Fractional Digits</td>
<td>Decimal</td>
<td>Self.content.FractionalDigits(5)</td>
</tr>
<tr>
<td>Minimum Inclusive</td>
<td>Decimal</td>
<td>Self.content.MinimumInclusive(20)</td>
</tr>
<tr>
<td>Maximum Inclusive</td>
<td>Decimal</td>
<td>Self.content.MaximumInclusive(25)</td>
</tr>
<tr>
<td>Minimum Exclusive</td>
<td>Decimal</td>
<td>Self.content.MinimumExclusive(30)</td>
</tr>
<tr>
<td>Maximum Exclusive</td>
<td>Decimal</td>
<td>Self.content.MaximumExclusive(35)</td>
</tr>
<tr>
<td>Minimum Inclusive Date</td>
<td>String</td>
<td>Self.content.MinimumInclusiveDate(yyyymmdd)</td>
</tr>
<tr>
<td>Maximum Inclusive Date</td>
<td>String</td>
<td>Self.content.MaximumInclusiveDate(yyyymmdd)</td>
</tr>
<tr>
<td>Minimum Exclusive Date</td>
<td>String</td>
<td>Self.content.MinimumExclusiveDate(yyyymmdd)</td>
</tr>
<tr>
<td>Maximum Exclusive Date</td>
<td>String</td>
<td>Self.content.MaximumExclusiveDate(yyyymmdd)</td>
</tr>
<tr>
<td>Expression</td>
<td>String</td>
<td>Regular Expression (non-OCL) that defines format restrictions. Eg \b[A-Z0-9.._%-]+@[A-Z0-9.-]+.[A-Z]{2,4}\b defines a regular expression for an e-mail address format.</td>
</tr>
</tbody>
</table>

Note:
CCTS has no primitive date type. Instead date is a Core Data Type (CDT) that includes a content component of primitive type string and supplementary components to define the date format. Therefore from the UML modeler perspective the date content component is a string and so the OCL contraints must act on the string representation.

### 6.6 Management Package – UML Profile

#### 6.6.1 Stereotypes and Tag Definitions (normative)

The following diagram shows a visualization of the Management Package to provide an overview of the defined elements. Note that the diagram shows only the structure of the UML Profile. Additional
constraints to define compliance with the semantics of the CCTS meta-model are provided in a subsequent section.

The sections below provide details on each stereotype in the Management Package. Both abstract and concrete stereotypes are described. However OCL constraints are only defined for concrete elements because these are the only ones that will exist in an actual instance model.

### 6.6.1.1 CC Library

<table>
<thead>
<tr>
<th>Stereotype</th>
<th>CCLibrary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Base Class</td>
<td>Package (from UML 1.4.2 / 2.0 meta-model)</td>
</tr>
<tr>
<td>Parent</td>
<td>Base::Management::BusinessLibrary</td>
</tr>
<tr>
<td>Description</td>
<td>The CCLibrary is a container for Core Components.</td>
</tr>
<tr>
<td>Tag Definitions</td>
<td></td>
</tr>
<tr>
<td>Tag Definition</td>
<td>Type</td>
</tr>
<tr>
<td>N/A</td>
<td></td>
</tr>
</tbody>
</table>

### 6.6.1.2 BIELibrary

<table>
<thead>
<tr>
<th>Stereotype</th>
<th>BIELibrary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Base Class</td>
<td>Package (from UML 1.4.2 / 2.0 meta-model)</td>
</tr>
<tr>
<td>Parent</td>
<td>Base::Management::BusinessLibrary</td>
</tr>
<tr>
<td>Description</td>
<td>The BIELibrary is a container for Business Information Entities (BIEs).</td>
</tr>
<tr>
<td>Tag Definitions</td>
<td></td>
</tr>
<tr>
<td>Tag Definition</td>
<td>Type</td>
</tr>
<tr>
<td>N/A</td>
<td></td>
</tr>
</tbody>
</table>
### 6.6.1.3 CDTLibrary

<table>
<thead>
<tr>
<th>Stereotype</th>
<th>CDTLibrary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Base Class</td>
<td>Package (from UML 1.4.2 / 2.0 meta-model)</td>
</tr>
<tr>
<td>Parent</td>
<td>Base::Management::BusinessLibrary</td>
</tr>
<tr>
<td>Description</td>
<td>The CDTLibrary is a container for the Core Data Types (CDT).</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Tag Definitions</th>
<th>Tag Definition</th>
<th>Type</th>
<th>Multiplicity</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>N/A</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### 6.6.1.4 QDTLibrary

<table>
<thead>
<tr>
<th>Stereotype</th>
<th>QDTLibrary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Base Class</td>
<td>Package (from UML 1.4.2 / 2.0 meta-model)</td>
</tr>
<tr>
<td>Parent</td>
<td>Base::Management::BusinessLibrary</td>
</tr>
<tr>
<td>Description</td>
<td>The QDTLibrary is a container for Qualified Data Types (QDT).</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Tag Definitions</th>
<th>Tag Definition</th>
<th>Type</th>
<th>Multiplicity</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>N/A</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### 6.6.1.5 DOCLibrary

<table>
<thead>
<tr>
<th>Stereotype</th>
<th>DOCLibrary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Base Class</td>
<td>Package (from UML 1.4.2 / 2.0 meta-model)</td>
</tr>
<tr>
<td>Parent</td>
<td>Base::Management::BusinessLibrary</td>
</tr>
<tr>
<td>Description</td>
<td>The DOCLibrary is a container for the assembled business document which is constructed from reusable ABIEs. The DOCLibrary can also contain ABIEs which are not reused elsewhere.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Tag Definitions</th>
<th>Tag Definition</th>
<th>Type</th>
<th>Multiplicity</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>N/A</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### 6.6.1.6 PRIMLibrary

<table>
<thead>
<tr>
<th>Stereotype</th>
<th>PRIMLibrary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Base Class</td>
<td>Package (from UML 1.4.2 / 2.0 meta-model)</td>
</tr>
<tr>
<td>Parent</td>
<td>Base::Management::BusinessLibrary</td>
</tr>
<tr>
<td>Description</td>
<td>The PRIMLibrary is a container for the UN/CEFACT primitive types.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Tag Definitions</th>
<th>Tag Definition</th>
<th>Type</th>
<th>Multiplicity</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>N/A</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
6.6.1.7 ENUMLibrary

<table>
<thead>
<tr>
<th>Stereotype</th>
<th>ENUMLibrary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Base Class</td>
<td>Package (from UML 1.4.2 / 2.0 meta-model)</td>
</tr>
<tr>
<td>Parent</td>
<td>Base::Management::BusinessLibrary</td>
</tr>
<tr>
<td>Description</td>
<td>The ENUMLibrary is a container for enumerated types which represent code lists.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Tag Definitions</th>
<th>Tag Definition</th>
<th>Type</th>
<th>Multiplicity</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>N/A</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

6.6.2 Constraints (normative)

A CDTLibrary shall only contain classes of stereotype <<CDT>>.

**UML 1.4**

context Package

inv CDTLibrary_onlyCDTs:

self.isCDTLibrary() implies self.contents->forAll(isCDT())

**UML 2.0**

context Package

inv CDTLibrary_onlyCDTs:

self.isCDTLibrary() implies self.ownedMember->forAll(isCDT())

A QDTLibrary shall only contain classes of stereotype <<QDT>>.

**UML 1.4**

context Package

inv QDTLibrary_onlyQDTs:

self.isQDTLibrary() implies self.contents->forAll(isQDT())

**UML 2.0**

context Package

inv QDTLibrary_onlyQDTs:

self.isQDTLibrary() implies self.ownedMember->forAll(isQDT())
A CCLibrary shall only contain classes of stereotype <<ACC>> and associations of stereotype <<ASCC>>.

UML 1.4
context Package
inv CCLibrary_onlyCCs:
  self.isCCLibrary() implies
  self.contents->select(content | content.isACC() or content.isASCC())->size()=
  self.contents->size()

UML 2.0
context Package
inv CCLibrary_onlyCCs:
  self.isCCLibrary() implies
  self.ownedMember->select(content | content.isACC() or content.isASCC())->size()=
  self.ownedMember->size()

A BIELibrary shall only contain classes of stereotype <<ABIE>> and associations of stereotype <<ASBIE>>.

UML 1.4
context Package
inv BIELibrary_onlyBIEs:
  self.isBIELibrary() implies
  self.contents->select(content | content.isABIE() or content.isASBIE())->size()=
  self.contents->size()

UML 2.0
context Package
inv BIELibrary_onlyBIEs:
  self.isBIELibrary() implies
  self.ownedMember->select(content | content.isABIE() or content.isASBIE())->size()=
  self.ownedMember->size()

A ENUMLibrary shall only contain classes of stereotype <<ENUM>>.

UML 1.4
context Package
inv ENUMLibrary_onlyENUMs:
  self.isENUMLibrary() implies
  self.contents->forAll(isENUM())

UML 2.0
context Package
inv ENUMLibrary_onlyENUMs:
  self.isENUMLibrary() implies
  self.ownedMember->forAll(isENUM())
A DOCLibrary shall only contain classes of stereotype <<ABIE>> and associations of stereotype <<ASBIE>>.

### UML 1.4

context Package

inv DOCLibrary_onlyDOCs:
  self.isDOCLibrary() implies 
  self.contents->select(content | content.isABIE() or content.isASBIE())->size()= 
  self.contents->size() 

### UML 2.0

context Package

inv DOCLibrary_onlyDOCs:
  self.isDOCLibrary() implies 
  self.ownedMember->select(content | content.isABIE() or content.isASBIE())->size()= 
  self.ownedMember->size() 

---

6.6.3 Packages & Namespace Conventions for libraries

The BCSS package structure has been designed such that one model package maps to one XML namespace in deployment.

The Library stereotype inherits from the registry object stereotype. The “baseURN” tagged value is optional for the registry object – and so is technically optional for the library stereotype. However it is strongly recommended that the BaseURN tagged value be present and accurate for all Library packages.

6.6.3.1 BIE Library Conventions

The name of a BIE library package can be any meaningful string but is recommended to be equal to the local name part of the baseURN. So for example, an ABIE library package might have baseURN=”http://www.eurofer.org/EuroferXML/Ordering”. The name of the package would be “EuroferXML/Ordering” and it would be located in a higher level package with name (and baseURN) equal to “http://www.eurofer.org/”. In this way, the URN of any package within a nested library can be read by concatenating package names.

For CEFACT BIE libraries, the BaseURN naming MUST follow the namespace rules defined by the ATG2 XML Naming and Design Rules Specification:

<cefact namespace>::<type>::<status>::<name>::<version> For example:

urn:un:unece:uncefact:data:standard:TBG1ReusableABIELibrary:1.0.1
6.6.3.2 **CC Library Conventions**

The core component library baseURN is defined by CEFACT and will always be the same (but with different version numbers). For example:

urn:un:unece:uncefact:data:standard:CoreComponentLibrary:0.8.2

6.7 **OCL methods used in the BCSS Profile**

---

1. **Global OCL definitions used in the BCSS standard**

   **UML 1.4**
   
   ```
   package Foundation::Core
   context ModelElement
   
   --Returns true if the element is a BCC
def:  
   let isBCC : Boolean =  
   self.hasStereotype('BCC') and  
   self.oclIsKindOf(Attribute)
   
   --Returns true if the element is a BBIE
def:  
   let isBBIE : Boolean =  
   self.hasStereotype('BBIE') and  
   self.oclIsKindOf(Attribute)
   
   --Returns true if the element is a CON
def:  
   let isCON : Boolean =  
   self.hasStereotype('CON') and  
   self.oclIsKindOf(Attribute)
   
   --Returns true if the element is a SUP
def:  
   let isSUP : Boolean =  
   self.hasStereotype('CON') and  
   self.oclIsKindOf(Attribute)
   
   --Returns true if the element is a ACC
def:  
   let isACC : Boolean =  
   self.hasStereotype('ACC') and  
   self.oclIsKindOf(Class)
   
   --Returns true if the element is a ABIE
def:  
   let isABIE : Boolean =  
   self.hasStereotype('ABIE') and  
   self.oclIsKindOf(Class)
   
   --Returns true if the element is an ASCC
def:  
   let isASCC : Boolean =  
   self.hasStereotype('ASCC') and  
   self.oclIsKindOf(Association)
   
   --Returns true if the element is an ASBIE
def:  
   let isASBIE : Boolean =
   ```
self.hasStereotype('ASBIE') and  
self.oclIsKindOf(Association)

--Returns true if the element is a CDT
def:
  let isCDT : Boolean =  
    self.oclIsKindOf(Class) and  
    self.hasStereotype('CDT')

--Returns true if the element is a QDT
def:
  let isQDT : Boolean =  
    self.oclIsKindOf(Class) and  
    self.hasStereotype('QDT')

--Returns true if the element is a PRIM
def:
  let isPRIM : Boolean =  
    self.oclIsKindOf(Class) and  
    self.hasStereotype('PRIM')

--Returns true if the element is an ENUM
def:
  let isENUM : Boolean =  
    self.oclIsKindOf(Class) and  
    self.hasStereotype('ENUM')

--Returns true if the package is a CDTLibrary
def:
  let isCDTLibrary : Boolean =  
    self.oclIsKindOf(Package) and  
    self.hasStereotype('CDTLibrary')

--Returns true if the package is a QDTLibrary
def:
  let isQDTLibrary : Boolean =  
    self.oclIsKindOf(Package) and  
    self.hasStereotype('QDTLibrary')

--Returns true if the package is a CCLibrary
def:
  let isCCLibrary : Boolean =  
    self.oclIsKindOf(Package) and  
    self.hasStereotype('CCLibrary')

--Returns true if the package is a BIELibrary
def:
  let isBIELibrary : Boolean =  
    self.oclIsKindOf(Package) and  
    self.hasStereotype('BIELibrary')

--Returns true if the package is a ENUMLibrary
def:
  let isENUMLibrary : Boolean =  
    self.oclIsKindOf(Package) and  
    self.hasStereotype('ENUMLibrary')

--Returns true if the package is a DOCLibrary
def:
  let isDOCLibrary : Boolean =  
    self.oclIsKindOf(Package) and  
    self.hasStereotype('DOCLibrary')
--Returns true if the element has a stereotype equal to the --passed string
def:
let hasStereotype(st : String) : Boolean =
    self.stereotype->select(self.name=st)->notEmpty()

--checks if a dependency is stereotyped as basedOn and leads to --an ACC
def:
let isBasedOnToACC : Boolean =
    self.oclIsKindOf(Dependency) and
    self.hasStereotype('basedOn') and
    self.oclAsType(Dependency).client->
    select(client | client.isACC())->size()=1

-- checks if a dependency is stereotyped as basedOn and leads --to a CDT
def:
let isBasedOnToCDT : Boolean =
    self.oclIsKindOf(Dependency) and
    self.hasStereotype('basedOn') and
    self.oclAsType(Dependency).client->
    select(client | client.isCDT())->size()=1

-- returns true if the association type is "Composition"
def:
let isComposition() : Boolean =
    self.oclIsKindOf(AssociationEnd) and
    self.oclAsType(AssociationEnd).aggregation =
    AggregationKind::composite

endpackage

UML 2.0
package Classes::Kernel
context Element
    --Returns true if the element is a BCC
def:
    let isBCC : Boolean =
        self.hasStereotype('BCC') and
        self.oclIsKindOf(Property)

    --Returns true if the element is a BBIE
def:
    let isBBIE : Boolean =
        self.hasStereotype('BBIE') and
        self.oclIsKindOf(Property)

    --Returns true if the element is a CON
def:
    let isCON : Boolean =
        self.hasStereotype('CON') and
        self.oclIsKindOf(Property)

    --Returns true if the element is a SUP
def:
    let isSUP : Boolean =
        self.hasStereotype('CON') and
        self.oclIsKindOf(Property)
--Returns true if the element is a ACC
def:
  let isACC : Boolean =
      self.hasStereotype('ACC') and
      self.oclIsKindOf(Class)

--Returns true if the element is a ABIE
def:
  let isABIE : Boolean =
      self.hasStereotype('ABIE') and
      self.oclIsKindOf(Class)

--Returns true if the element is an ASCC
def:
  let isASCC : Boolean =
      self.hasStereotype('ASCC') and
      self.oclIsKindOf(Association)

--Returns true if the element is an ASBIE
def:
  let isASBIE : Boolean =
      self.hasStereotype('ASBIE') and
      self.oclIsKindOf(Association)

--Returns true if the element is a CDT
def:
  let isCDT : Boolean =
      self.oclIsKindOf(Class) and
      self.hasStereotype('CDT')

--Returns true if the element is a QDT
def:
  let isQDT : Boolean =
      self.oclIsKindOf(Class) and
      self.hasStereotype('QDT')

--Returns true if the element is a PRIM
def:
  let isPRIM : Boolean =
      self.oclIsKindOf(Class) and
      self.hasStereotype('PRIM')

--Returns true if the element is an ENUM
def:
  let isENUM : Boolean =
      self.oclIsKindOf(Class) and
      self.hasStereotype('ENUM')

--Returns true if the package is a CDTLibrary
def:
  let isCDTLibrary : Boolean =
      self.oclIsKindOf(Package) and
      self.hasStereotype('CDTLibrary')

--Returns true if the package is a QDTLibrary
def:
  let isQDTLibrary : Boolean =
      self.oclIsKindOf(Package) and
      self.hasStereotype('QDTLibrary')
--Returns true if the package is a CCLibrary
def:
    let isCCLibrary : Boolean =
        self.oclIsKindOf(Package) and
        self.hasStereotype('CCLibrary')

--Returns true if the package is a BIELibrary
def:
    let isBIELibrary : Boolean =
        self.oclIsKindOf(Package) and
        self.hasStereotype('BIELibrary')

--Returns true if the package is a ENUMLibrary
def:
    let isENUMLibrary : Boolean =
        self.oclIsKindOf(Package) and
        self.hasStereotype('ENUMLibrary')

--Returns true if the package is a DOCLibrary
def:
    let isDOCLibrary : Boolean =
        self.oclIsKindOf(Package) and
        self.hasStereotype('DOCLibrary')

--Returns true if the element has a stereotype equal to the passed string
def:
    let hasStereotype(st : String) : Boolean =
        self.stereotype->select(self.name=st)->notEmpty()

-- checks if a dependency is stereotyped as basedOn and leads to an ACC
def:
    let isBasedOnToACC : Boolean =
        self.oclIsKindOf(Dependency) and
        self.hasStereotype('basedOn') and
        self.oclAsType(Dependency).client->select(client | client.isACC())->size()=1

-- checks if a dependency is stereotyped as basedOn and leads to a CDT
def:
    let isBasedOnToCDT : Boolean =
        self.oclIsKindOf(Dependency) and
        self.hasStereotype('basedOn') and
        self.oclAsType(Dependency).client->select(client | client.isCDT())->size()=1

-- returns true if the association type is "Composition"
def:
    let isComposition() : Boolean =
        self.oclIsKindOf(Property) and
        self.oclAsType(Property).aggregation =
        AggregationKind::composite
6.8 Context Package – UML Profile

UML based concepts to represent context will be added based on the results of the Unified Context Methodology (UCM) project from TMG which is currently in ODP step 2.

6.9 Relationship to UMM

The UMM Business Transaction View package contains sub packages of type <<BusinessInformationView>>. This package should contain only a class of stereotype <<InformationEnvelope>> element that has an association to an ABIE in a <<DOCLibrary>> package. The <<DocLibrary>> package, in turn contains one or more assembly documents that are constructed from ABIEs in packages of stereotype <<ABIELibrary>>

This approach is designed to separate three key namespaces:

- The namespace of the UMM Business process (typically realized in a BPSS, WS-CDL or abstract BPEL schema).
- The namespace of the Business Information that is used in the business process (typically realised as an XSD schema)
- The namespace of the ABIE library components that are used to build the business information (typically realised as re-usable XSD schema that are imported into the root schema)

7 Example

This section is informative. It is provided as a guide to the business modeller on best practices for CCTS modelling using this UML profile, illustrated with examples.

7.1 Structure

An information model based on the UML profile for CCTS can contain UML packages with the stereotypes “DocLibrary”, “BIELibrary”, “CCLibrary”, “QDTLibrary”, “CDTLibrary”, “ENUMLibrary” and “PRIMLibrary”. These packages can be further structured within packages with stereotype “BusinessLibrary”. The following figure shows the packages of an example model based on the UML profile for CCTS. This example will be explained in the subsequent sections.
7.2 PRIMLibrary

The package with the stereotype “PRIMLibrary” contains the fixed set of CEFACT primitive types as defined in the CCTS. The CEFACT primitive types are later on used to provide primitive types for the content components (CON) and supplementary components (SUP) of the core data types (CDT) and qualified data types (QDT) and are represented as UML classes with the stereotype “PRIM”:

Usage

The PRIMLibrary will be provided as part of the final version of this technical specification as an XMI file to be imported into a given UML tool.

7.3 ENUMLibrary

The package with the stereotype “ENUMLibrary” contains the enumerations (code lists) that are used within the model. Enumerations may later on used to restrict the content component (CON) of qualified data types (QDT) and are represented as UML classes with the stereotype “enumeration”. Note that enumerations can also be referenced externally as part of qualified data types and therefore don’t need to be part of a model.

Usage

ENUMLibraries can be imported as an XMI file into a given UML tool or hand-crafted as part of a modeling project.

---

1 A final solution to message assembly will be provided by the Core Components Message Assembly Project (CCMA) project which is currently in ODP step 3.
### 7.4 CDTLibrary

The package with the stereotype “CDTLibrary” contains the fixed set of core component types as defined in the CCTS. The core component types are later on used to derive qualified data types (QDT) by restriction and are represented as UML classes with the stereotype “CDT” containing exactly one content component as a UML attribute with the stereotype “CON” and zero to many supplementary components as UML attributes with the stereotype “SUP”. Dependencies between a CDTLibrary and other Libraries can be expressed using the standard UML dependency.

#### Usage

The CDTLibrary will be provided as part of the final version of this technical specification as an XMI file to be imported into a given UML tool.

### 7.5 QDTLibrary

The package with the stereotype “QDTLibrary” contains the qualified data types (QDT) used within the model. The qualified data types are derived by restriction from CDTs and are represented as UML classes with the stereotype “QDT” containing exactly one content component as a UML attribute with the stereotype “CON” and zero to many supplementary components as UML attributes with the stereotype “SUP”. Dependencies between a QDTLibrary and other Libraries can be expressed using the standard UML dependency.
The fact that a QDT is derived from a CDT is represented as a UML dependency with stereotype “basedOn” between a QDT and a CDT. The name of a QDT contains the name of the CDT it is based on prefixed by one to many semantic qualifiers separated with underscore and blank as defined in the CCTS:

Note that the qualified data types in the example are restricted by means of:
- restriction of the content component (CON) by assigning an enumeration from the “ENULibrary”
- restriction of the content component (CON) by assigning a written constraint
- removing supplementary components (SUP)
- assigning default values to supplementary components (SUP)

Usage
QDTLibraries can be imported as an XMI file into a given UML tool or hand-crafted as part of a modeling project (as candidate QDTs).

If there is no additional support from the UML tool in use, a QDT is created by:
- copy (duplicate) the (to be restricted) CDT into a package with stereotype “QDTLibrary”,
- change the stereotype from CDT to QDT,
- draw a dependency with the stereotype “basedOn” between the QDT and the CDT it is based on and
- define the QDT by restricting it and adding semantic qualifiers.
7.6 CCLibrary

The package with the stereotype “CCLibrary” contains aggregate core components represented as UML classed with stereotype “ACC” consisting of basic core components represented as UML attributes with stereotype “BCC” and association core components represented as UML compositions with stereotype “ASCC”. Aggregate core components are generic information objects as defined in the CCTS. The aggregate core components (ACC) are later on used to derive aggregate business information entities (ABIE) by restriction. The name of a class with stereotype “ACC” is the object class term as defined in the CCTS while the names the attributes with stereotype “BCC” are the property terms and the target roles of compositions with stereotype “ASCC” are the representation terms (not the dictionary entry names respectively). The types of the attributes with stereotype “BCC” are taken from the “CDTLibrary”. Dependencies between a CCLibrary and other Libraries can be expressed using the standard UML dependency. The given example uses the core components library published by TBG17:

Usage

The CCLibrary can be imported as an XMI file into a given UML tool.

7.7 BIELibrary

The package with the stereotype “BIELibrary” contains aggregate business information entities represented as UML classed with stereotype “ABIE” consisting of basic business information entities represented as UML attributes with stereotype “BBIE” and association business information entities represented as UML compositions with stereotype “ASBIE”. Dependencies between a BIELibrary and other Libraries can be expressed using the standard UML dependency.

ABIEs are reusable information objects that are derived by restriction from ACCs. The fact that an ABIE is derived from an ACC is represented as a UML dependency with stereotype “basedOn” between an ABIE and an ACC:
The names of ABIEs, BBIEs and ASBIEs contain the name of the respective ACC, BCC and ASCC they are based on prefixed by one to many semantic qualifiers separated with underscore and blank as defined in the CCTS.

Note that the classes with stereotype “ABIE” can be restricted by means of:

- restricted number of attributes (BBIEs) and compositions (ASBIEs)
- restricted multiplicity of attributes (BBIEs) and compositions (ASBIEs)
- restricted data types using qualified data types (QDT) as restricted CDTs to type attributes (BBIEs)
- restriction by assigning UML constraints

Usage

BIELibraries can be imported as an XMI file into a given UML tool or hand-crafted as part of a modeling project (candidate ABIEs as defined in the CCTS).

If there is no additional support from the UML tool in use, an ABIE is created by

- copy (duplicate) the (to be restricted) ACC into a package with stereotype “ABIELibrary”,

```xml
<ACC>
  + identification: Identifier [0..unbounded]
  + name: Text [0..unbounded]
</ACC>

<BBIE>
  + Age: Measure [0..unbounded]
  + Alias: Text [0..unbounded]
  + Birth: DateTime [0..unbounded]
  + Death: DateTime [0..unbounded]
  + Description: Text [0..unbounded]
  + Family Name: Text [0..unbounded]
  + Family Name Prefix: Text [0..unbounded]
  + Gender: Code [0..unbounded]
  + Given Name: Text [0..unbounded]
  + Identification: Identifier [0..unbounded]
  + Language: Identifier [0..unbounded]
  + Maiden Name: Text [0..unbounded]
  + Mental Status: Code [0..unbounded]
  + Middle Name: Text [0..unbounded]
  + Name: Text [0..unbounded]
  + Name Suffix: Text [0..unbounded]
  + Preferred Name: Text [0..unbounded]
  + Salutation: Text [0..unbounded]
  + Title: Text [0..unbounded]
</BBIE>
```
• define the ABIE by restricting it (e.g. set multiplicity by defining a single UML Multiplicity Range for the attribute, change the data type by referenced by assigning the type of the attribute to a class stereotyped with “QDT”),
• add semantic qualifiers,
• change the remaining stereotypes from ACC to ABIE, from BCC to BBIE and from ASCC to ASBIE and
• draw a dependency with the stereotype “basedOn” between the ABIE and the ACC it is based on.

7.8 DocLibrary

The package with the stereotype “DocLibrary” contains the business information assembled from reusable ABIEs to be exchanged in a given business scenario (the exact link between CCTS and message assembly will be clarified by the CCMA project). Dependencies between a DocLibrary and other Libraries can be expressed using the standard UML dependency.

7.9 Proposed link to UMM artefacts (UMM Foundation 1.0 – Final Working draft)

Business information from a “DOCLibrary” can be used as part of an UMM collaboration model within the “BusinessInformationView” (see UMM Foundation Module 1.0 for further information):
8 Model Interchange based on XMI / proposed XMI flavor

The XMI-specification is a procedure which defines how an instance of an arbitrary MOF meta model is to be stored in an XML representation. Thus XMI varies with the MOF version, the UML version and the version of the XMI standard used. All combinations of these factors represent legal and practiced versions of UML XMI. Considering two MOF versions of the 1.x variety, potentially four UML Versions of the 1.x variety and two revisions of the XMI standard, we may expect 16 different legal types of XMI. To complicate matters further, the XMI standard is ambiguous and leaves room for interpretation.

In order to achieve interoperability there must be a point of reference to define a notion of correctness for XMI-encoded UML and thus UMM. One way to solve this problem is to define constraints on XMI using an XML-based mechanism like DTD, Schema or relax-ng and publish these constraints as a "XMI-Profile", so vendors can implement this XMI-flavour. It is important to note, that this approach defines one specific combination of UML, XMI and MOF. By this fact, it might already exclude certain tools from interoperability. Further, if the underlying UML is restricted to only cover a certain area of the UML (e.g. class diagrams), as has been done in other specifications, implicit semantic constraints on the language are implicitly introduced, which is explicitly prohibited by the OMG document defining the use of profiles. To comply with an approach along the lines with such specifications, vendors are forced to write transformations and filters to alter there XMI in this very specific manner. It is not certain that vendors will be inclined to implement such a complex and costly transformation for a small market. (This approximately equates to requiring vendors to implement different dialects of an EDIFACT defined standard for each community.)

Another way to define validity of XMI is to refer to an existing XMI-flavour - one existing solution comes from the Java Community Process (JCP) that provides an interface called the Java Metadata Interface (JMI) as part of JSR-40 (http://www.jcp.org/en/jsr/detail?id=40) that reads a given XMI-file. As part of this process, a technical compliance suite has been provided, which can be used to test modelling tools XMI exports. Other solutions are to use the Eclipse UML2-project as a reference or refer to the rules to be established by the UML 2.0 Tools Certification program of the OMG (http://www.omg.org/xmitest/).

The requirement to comply with either of these reference XMI flavors can be regarded as sufficient to guarantee XMI-interoperability.

The feasibility of these approaches will be evaluated as part of the implementation verification phase of this project.

9 Conformance Criteria

This section is normative.

Conformance statements conform to the requirements of the W3C QA guidelines:

http://www.w3.org/TR/2004/WD-qaframe-spec-20041122/

9.1 Product Classes

The following types of product may claim conformance with this specification:

- UML Modelling tools from tool vendors may claim compliance
- UML Model instances from working groups (eg TBG) may claim compliance
- External Validation services (eg ICG repository service) may claim compliance

9.2 Profiles and Levels

The W3C QA guidelines allow the specification of different level of compliance. For BCSS this might mean compliance only with the structural components (stereotypes & tagged values) or compliance with
the complete specification including the semantic constraints defined by the OCL constraints. It is proposed to delay definition of compliance levels until implementation verification phase.

10 Reference Implementation

A reference implementation will be provided as part of the implementation verification phase of this project and will include:

- A validation service to validate models claiming conformance.
- A test scenario including at least three UML tools and the UN/CEFACT registry/repository from ICG to prove the feasibility of the exchange of models based on the proposed XMI flavor (MDR/JMI).
- A service to generate ATG NDR compliant deployment schema from models represented in XMI.
- Example models.

Experiences from the implementation verification phase will be used to advance this specification regarding conformance, validation, storage, management and other aspects improving interchange of core component models based on the UML Profile for CCTS 2.01.