A UML Profile for Core Components and their Transformation to XSD

Christian Huemer
Institute of Software Technology
Vienna University of Technology
Favoritenstr. 9-11/188, 1040 Vienna, Austria
huemer@big.tuwien.ac.at

Philipp Liegl
Research Studios Austria
Austrian Research Centers GmbH - ARC
Thurngasse 8/20, 1090 Vienna, Austria
pliegl@researchstudio.at

Abstract

In business-to-business e-commerce, traditional electronic data interchange (EDI) approaches such as UN/EDIFACT have been superseded by approaches like web services and ebXML. Nevertheless, a precise and common semantic definition of business documents exchanged is needed. In order to become independent from a transfer syntax, we prefer defining the documents as platform independent models. An approach that follows this idea is the UN/CEFACT's core component standard. Core components are reusable semantic building blocks which can be combined in various ways to create shared libraries of interoperable business documents. In order to use standard UML modeling tools we have developed a UML profile for the core components standard. Furthermore, we adapt the UN/CEFACT naming and design rules for the UML profile in order to derive XML schemas for business document exchanges. The overall approach is demonstrated by using a specific example from the field of e-commerce.

1 Motivation

The idea of harmonizing data exchanged between two business partners is not new and was adopted by a set of standardization bodies. It became known as electronic data interchange (EDI). Probably the best known standard is UN/EDIFACT [11], maintained by the United Nations Centre for Trade Facilitation and Electronic Business (UN/CEFACT). In the late 1990ies XML became the defacto standard for exchanging data over the Internet. As a result, various standards organizations developed standard business document types based on XML. An overview of different XML-based business document standards is presented by Li [5].

Although an XML-based approach seemed to be very promising at first sight, it soon became clear that the move to XML will not solve all the interoperability problems in business document exchanges. First of all, a multitude of different and competing XML-based document standards were developed that had no common ontological base. As a consequence, one had to implement either many different standards or to stick to a limited user base. Furthermore, the standardization approach of XML-based standards was similar to the one used in EDI. It is the attempt to include every possible element that may be required in any partnership, even knowing that it is not used in most of the partnerships. This results in overloaded and highly optional document structures of which only about 3% are used in a particular partnership. Accordingly, partners have to agree on a message implementation guideline cutting down the standard to their own needs before implementing the inter-organizational system. When XML appeared, another drawback was realized by designers and implementers of UN/EDIFACT: the uncertainty of future developments. If the semantics of a business document type do not change, it is ineffective to start the standardization effort for each transfer syntax again.

Knowing these limitations UN/CEFACT - which we are a member of - had the idea to develop an ontological base of re-useable building blocks for creating shared libraries of interoperable business documents. This base is built upon the most complete business ontology that exists today, i.e. the United Nations Trade Data Element Dictionary (UN/TDED). The UN/CEFACT effort became known as core components. It was initially started as part of the ebXML standards suite [9]. The current version of the core components standard released by UN/CEFACT is 2.1 [13] with 3.0 under development. Today, core components are not seen as an approach limited to ebXML. Rather they are a platform independent data modeling approach for business documents. Naming and design rules will be used to transform the models to a specific transfer syntax. In this paper we demonstrate an implementation of the core components UML profile in the UML modeling tool Enterprise Architect. Furthermore a tool-supported transformation of UML based core component definitions to XML schemas

1-4244-0832-6/07/$20.00

is presented. The generated schemas [16] may be used in ebXML as well as Web Service platforms

The core components standard is based on its own metamodel. This resulted in some serious drawbacks for practical use. Tool support for core components modeling is very limited and there is no format defined to register and exchange core components. Accordingly, the standardization and harmonization process of core component instances is based on spread sheets. In order to overcome these limitations we started a project within UN/CEFACT developing a UML profile for core components. Thereby, we hope to gain better tool support and to use XMI for registering and exchanging core components. It follows that the transformation to XML schemas is started from UML class diagrams. Today we know many different approaches to map UML to XML, e.g. [1], [2], [3], [4], [10], [6]. Our approach differs from these ones by being based on the ontological basis of core components, by a well-defined mechanism to generate schemas from different business libraries and by incorporating the UN/CEFACT naming and design rules.

The remainder of the paper is structured as follows: Section 2 gives an overview of the core components standard. The UML profile for core components which we have developed is presented in Section 3. Section 4 describes the production process of XML schemas when applying UN/CEFACT naming and design rules to our UML profile. Section 5 describes future work and concludes the paper.

2 The core components technical specification

Core components are the central building blocks of the CCTS standard. The CCTS standard distinguishes between three different types of core components namely *basic core components* (BCC), *aggregate core components* (ACC) and *association core components* (ASCC). *Basic core components* are atomic values such as street or postal code. They are consolidated in so called *aggregate core components*. An *aggregate core component* consists of several *basic core components* (e.g. an address). In order to model dependencies between different *aggregate core components* so called *association core components* can be used. Such a dependency could for instance exist between a person and an address. In order to give a first impression about how core components are represented in UML, we use a simple example as depicted in figure 1. In UML *aggregate core components* are represented by classes, *basic core components* by class attributes and *association core components* by associations and compositions respectively.

2.1 Core Components

The left hand side of figure 1 shows two sample core components (CC) *Person* and *Address*, both of type *aggregate core component* (ACC). An aggregate core component can contain a set of information fields, so called *basic core components* (BCC) associated with certain data types. In the aggregate core component *Person* the BCC *DateofBirth* is of type *Date* and *FirstName* is of type *Text*. Hence, aggregate core components can be regarded as a collection of related pieces of business information, forming a distinct business meaning. The aggregate core component *Person* has two *association core components* (ASCC) namely *Private* and *Work*, both of type *Address*. Similar to the concept pursued in object orientation, the two association core components *Work* and *Private* will become attributes of the aggregate core component *Person* once the model is transferred into code. Association core components therefore are nothing more than basic core components representing a *complex type* - in this case *Address*. The aggregate core component *Person* shown on the left hand side of figure 1 will therefore result in the following set of core components: *Person* (ACC), *Person.DateofBirth* (BCC), *Person.FirstName* (BCC), *Person.Private.Address* (ASCC), *Person.Work.Address* (ASCC).

The data types used for basic core components are so called *core data types* (CDT) representing the full range of values that shall be used for the representation of a particular property. By definition, core data types do not have a business semantic.

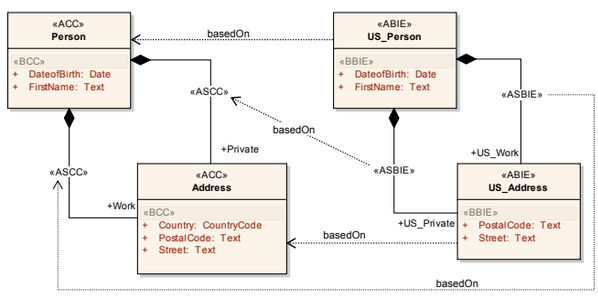

Figure 1. Dependency between Core Components and Business Information Entities

2.2 Business Information Entities

By introducing the business context, we can qualify and refine core components according to the needs of a specific industry or domain. Refined core components, used in a specific context are referred to as *business information entities*. *Context* in this case can for instance be *travel industry*

or *chemical industry*. An address in the first context for instance differs from an address in second context - hence a core component *address* cannot be used in both context. However, by deriving business information entities from the core component *address* the user has the possibility to use a tailored core component *address* for every specific context. Therefore a *business information entity* (BIE) is a restricted *core component* and represents a piece or a set of business data having a unique business semantic meaning.

The CCTS standard distinguishes between three different business information entities, namely *basic business information entities* (BBIE), *aggregate business information entities* (ABIE) and *association business information entities* (ASBIE). Akin to core components there are dependencies between the different business information entity types as well. *Basic business information entities* represent atomic values and are consolidated in *aggregate business information entities*. The dependencies between different *aggregate business information entities* are modeled by using *association business information entities*. The right hand side of figure 1 shows two sample aggregate business information entities *US_Person* and *US_Address*. Aggregate business information entities are a collection of related basic business information entities, conveying a certain business meaning in a specific business context. A basic business information entity represents a simple business information. The aggregate business information entity *US_Address* has two BBIEs namely *PostalCode* and *Street*. Like a basic core component, a basic business information entity has a certain data type. The data type of a basic business information entity can either be a *core data type* (CDT) or a *qualified data type* (QDT). Data types will be explained in detail in section 3. Similar to the concept of association core components, association business information entities are attributes of an aggregation business information entity, representing another aggregation business information entity. In figure 1 the aggregate business information entity *US_Person* has two association business information entities namely *US_Private* and *US_Work*. When the business information entities are transferred into code, the ASBIEs *US_Private* and *US_Work* will become attributes of the ABIE *US_Person* and are of type *US_Address*. The business information entity *US_Person* will therefore result in the following set of business information entities: *US_Person* (ABIE), *US_Person.DateofBirth* (BBIE), *US_Person.FirstName* (BBIE), *US_Person.US_Private.US_Address* (ASBIE), *US_Person.US_Work.US_Address* (ASBIE).

2.3 The Core Component Meta Model

The interdependencies between core components and business information entities are specified in the core components meta model. As already mentioned, we distinguish between a core context and a business specific context whereas the elements of the business context depend on the underlying core elements. Figure 2 shows the dependencies between the used model elements.

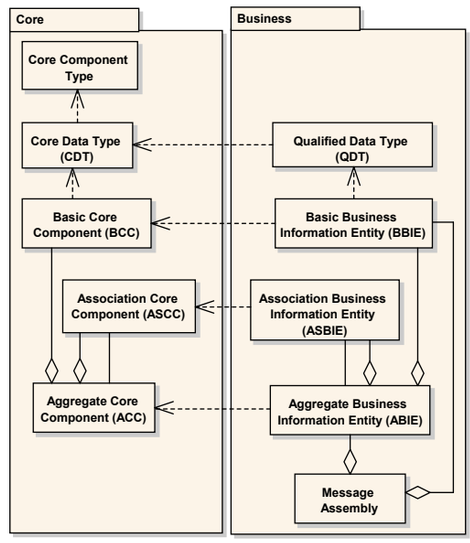

Figure 2. The core components meta model

For the sake of simplicity the generic terms *core component* and *business information entity* are used instead of the specific terms. Where semantic differentiation prescribes it, the specific term will be used.

2.3.1 Deriving business information entities

ABIEs are exclusively derived from ACCs by restriction. The two ABIEs *US_Address* and *US_Person* as shown in figure 1 are based on the core components *Person* and *Address*. This is denoted by the *basedOn* dependency. Please note that *US_Address* is missing the attribute *Country*, hence the core component *Address* was restricted in order to create the new business information entity *US_Address*. Association core components and association business information entities are related in the same manner as ACCs and ABIEs - ASBIEs depend on ASCCs. In figure 1 this relationship is also denoted by a *basedOn* dependency. The specific business context of an ABIE and an ASBIE is shown by adding an optional prefix to the name of the underlying core component. In figure 1 the prefix *US_* is used.

3 A UML profile for core components

For the rest of the paper the model depicted in figure 4 will serve as a reference for the explanations given. The

left hand side of figure 4 shows a so called tree view, where all the packages located in the model are shown. For seven specific packages the relevant diagrams are shown on the right hand side of figure 4. Compelled by space limitations, some elements are shown in the tree view, but not in the corresponding diagram. Due to the transfer syntax independence of core components, basically any format of representation can be used. In order to allow the modeling of core components with an appropriate UML tool, it was necessary to describe the core components meta model using a UML profile. The UML profile for core components defines a set of stereotypes, tagged values and OCL constraints and has exemplarily been implemented as a toolbar for the UML modeling tool Enterprise Architect.

In 2005 we started the work on the *UML Profile for Core Components based on CCTS 2.01* within the TMG group of UN/CEFACT. In October 2006 the Candidate for version 1.0 was released [14]. Figure 3 gives an overview about the different stereotypes used in the profile. The profile con-

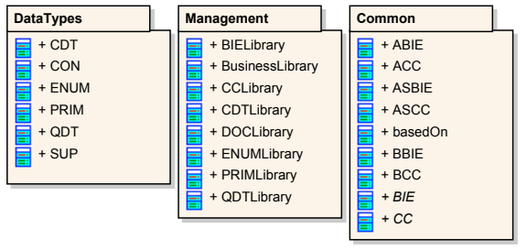

Figure 3. A UML profile for core components

sists of eight libraries located in the *Management* package, six data types located in the *DataTypes* package and nine stereotypes located in the *Common* package used to model core components, business information entities and their relationships.

The stereotypes in the *Common* package are to be read according to the abbreviations given in section 2.1 and 2.2.

The data types used are located in the *DataTypes* package. A *core data type* (CDT) is a complex data type according to the approved *Core Component Types* listed in the CCTS standard [13] e.g. *DateTime*. Core data types consist of exactly one attribute stereotyped as *CON* and zero or more attributes stereotyped as *SUP*. CON refers to *content component* and SUP to *supplementary component*. The content component element carries the actual content of the core data type whereas the supplementary components provide additional meaning to the content component. In package 4 on the left hand side of figure 4 four core data types are shown namely *Code*, *Identifier*, *Text* and *Name*. Due to space limitations only *Code* is shown in detail in the diagram of package 4. As shown in package 4, the core data type *Code* has exactly one content component named *Content*. Its data type is *String*. Additionally *Code* has four supplementary components describing the content component. The four supplementary components are *CodeListAgName*, *CodeListName*, *CodeListSchemeURI* and *LanguageIdentifier*. Hence, supplementary components can be regarded as meta information about the content component.

Like business information entities are created from core components, *qualified data types* (QDT) are created from core data types by restriction. The relationship between a core data type and a qualified data type is shown by a dependency stereotyped as *basedOn*. In package 3 of figure 4 the qualified data types *CountryType* and *CouncilType* are shown. Both are derived from the core data type *Code* which is indicated by the *basedOn* dependency between the types. The core data type *Code* is originally defined in package 4 and has only been drawn in package 3 to show the dependency between CDT and QDT.

A qualified data type can contain supplementary components and must contain exactly one content component. The supplementary components of a qualified data type are derived from the supplementary components of the underlying core data type by restriction. As depicted in the diagram of package 3, the core data type *Code* contains four supplementary components. However, only the supplementary component *CodeListName* is actually used in the qualified data types *CountryType* and *CouncilType* derived from *Code* due to the derivation-by-restriction mechanism.

One important aspect of a qualified data type is the type of the content component and the supplementary components. Content component and supplementary components of a qualified data type can be restricted to a specific set of values by assigning an *enumeration type* (ENUM) to it. The content components of the two qualified data types *CountryType* and *CouncilType* shown in package 3 of figure 4 are restricted by enumerations. The content component of *CountryType* is of type *CountryType_Code* and the content component of *CouncilType* is of type *CouncilType_Code*. ENUM elements are defined in so called *ENUMLibraries*. Package 6 of figure 4 shows two enumeration types *CouncilType_Code* and *CountryType_Code*.

A *primitive type* (PRIM) represents one of the primitive types as defined in the CCTS standard e.g. *Integer*. In package 7 of figure 4 three primitive types are depicted.

Within the *Management* package of the UML profile for core components shown in figure 3, the different containers for the data types are defined. Each library contains a specific data type as described in the *DataType* package. The libraries are used to logically group the different element types and facilitate their reuse in the model.

Denoted by package 1 in figure 4 a *DOCLibrary* is shown. *DOCLibraries* contain association business information entities imported from other packages which are assembled to a new business document. Literally spoken,

a *DOCLibrary* therefore represents a final business document. In package 1 of figure 4 a *DOCLibrary* named *HoardingPermit* is shown. In addition to its BBIEs, *HoardingPermit* has four ASBIEs namely *Included* (leading to *Attachment*), *Current* (leading to *Application*), *Billing* (leading to *Person_Identification*) and *Included* (leading to *Registration*). In the *DOCLibrary* itself two aggregate business information entities are defined, namely *HoardingDetails* and *HoardingPermit*. The latter one is not used in the actual diagram. The association business information entities are all coming from different *BIELibraries* and are assembled to the business document *HoardingPermit* in the *DOCLibrary*.

As an example the *BIELibrary CommonAggregates* is shown in package 2. It defines four different aggregate business information entities namely *Person_Identification*, *Signature*, *Address* and *Application*. As already explained, an aggregate business information entity is derived from an aggregate core component by restriction. The aggregate business information entity *Application* is based on the aggregate core component *Application* defined in the *CCLibrary* depicted in package 5. Of the initially eleven basic core components of the aggregate core component *Application*, only two are actually used as basic business information entities (*CreatedDate* and *Type*) in the aggregate business information entity *Application*, the others are omitted due to the derivation-by-restriction mechanism. The only purpose of a *BIELibrary* is to define the interdependencies between business information entities and provide them for reuse in the *DOCLibraries*. In package 2 of figure 4 the basic business information entity *CountryName* of the aggregate business information entity *Address* is of type *CountryType* - a qualified data type.

Package 3 shows the *QDTLibrary CommonDataTypes* and its two qualified data types *CountryType* and *CouncilType*. Both data types are based on the same core data type *Code* and both content components are restricted by enumerations. The *CDTLibrary* in which the core data type *Code* is located is shown in package 4. Due to space limitations the three other core data types *Identifier*, *Text* and *Name* are not shown in the diagram.

In package 6 an *ENUMLibrary* is presented, containing the enumerations used in the qualified data types *CountryType* and *CouncilType*. Package 7 shows a *PRIMLibrary* containing three primitive types *String*, *Boolean* and *Integer*.

Different libraries containing data types are aggregated into so called *business libraries*. As shown on the left hand side of figure 4, a core components model can contain multiple business libraries.

4 Deriving XSD schemas

The generation of XML schemas from a core components model is shown on the basis of the model described in figure 4.

With the use of the UML profile, a unique and unambiguous representation mechanism is available for core components. As already explained, the basic idea of core components is to assemble business documents from predefined building blocks. A UML diagram of core components, however, is only defining the business document on an abstract and conceptual level. What is needed, is the transfer to a normative document such as an XML schema, which can then be used to validate document instances against it. In our example we have chosen to derive XML schemas from the core components model. The schemas are then used to validate XML messages exchanged during a business process. Nevertheless the generation is not necessarily limited to XML schema and future extensions could include the generation of RELAX NG [8] or RDF schemas [15] as well.

Due to the huge amount of core components, business information entities etc. in a large model, a manual transformation to a schema is unmanageable. Therefore we have developed an automatic solution and implemented it in our UMM Add-In [7] of the UML modeling tool Enterprise Architect.

Additionally to the release of the CCTS [13] so called *Naming and Design Rules* (NDR) [12] have been released by the Applied Technology Group (ATG) of UN/CEFACT. The recommendations given in the NDR are reflected in our implementation.

4.1 A sample generation

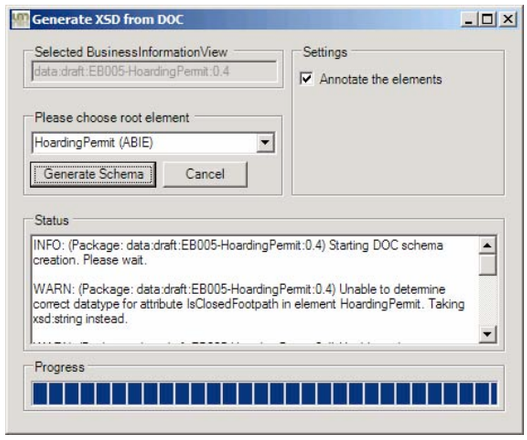

Figure 5. The XSD generator

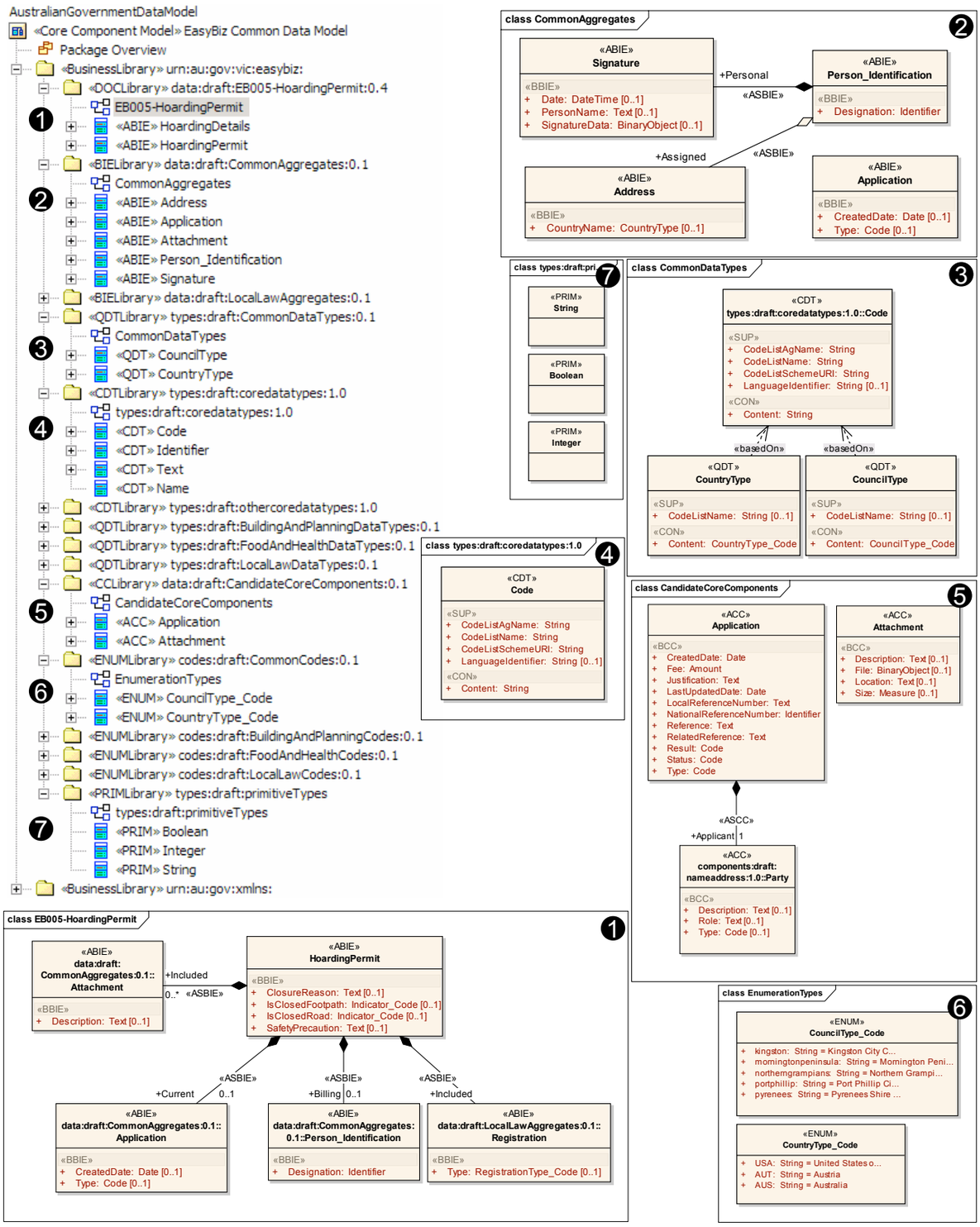

Figure 4. A CCTS example

Schemas can be generated separately for every sub-library described in the *Management* package of figure 2. Using a simple right-click on the library the user can open a generator dialog. In most of the cases the user may want to generate a schema from a *DOCLibrary*. Figure 5 shows the dialog window for a *DOCLibrary* schema generator. Because a *DOCLibrary* can contain many aggregate business information entities, the user must first select a root element for the schema. After pressing *Generate Schema* the user is presented a dialog for choosing the folder the generated schemas will be saved in. During the generation of the schema, status messages are passed back to the user interface. In case the UML model is erroneous, the generation aborts and the user is presented an error message.

Generating from a DOCLibrary

In a selected *DOCLibrary* the Add-In starts at the selected root element and pursues every outgoing aggregation and composition connector. Interdependencies to other libraries are evaluated and the necessary schemas are generated. Figure 6 shows the XSD schema which has been created for the *DOCLibrary HoardingPermit* depicted in package 1 of figure 4. In line 1 of figure 6 the necessary namespaces for the imported schemas are defined. Relevant schemas are automatically generated and imported for every element defined in a different package and used in the *DOCLibrary*. The schema imports are shown in the lines 2 - 5. Four different schemas are generated and imported. In line 2 a schema defining the core data types used by the ABIEs in the *DOCLibrary* is imported. Line 3 imports a schema defining the qualified data types used by the ABIEs in the *DOCLibrary*. And finally line 4 and 5 import two *BIELibraries* in which the additional ABIEs are defined for the use in the *DOCLibrary*. For the selected root element *HoardingPermit* exactly one element is defined in line 18. The root element is of type *doc:HoardingPermitType* defined in line 6.

For every aggregate business information entity a *complexType* is defined which is named after the business entity plus a *Type* postfix. A *complexType* for an aggregate business information entity is defined by a *sequence* consisting of the basic business information entities and association business information entities the ABIE contains. As depicted in line 6, first the elements for the BBIEs are defined. The data types and the multiplicities are taken according to the definition in the UML model and transferred into the XML schema. The data type of the BBIEs *ClosureReason* and *SafteyPrecaution* defined in line 8 and 11 of figure 6 is of type *core data type*. This can easily be seen by examining the namespace assigned to the prefix *cdt1*. Hence the type definition of the XML element points to the right element in the imported schema. Per contra the data types of the BBIEs *IsClosedFootpath* and *IsClosedRoad* are of type *qualified data type*. Analogically to core data types, the type definitions of the two elements point to the right elements in the imported qualified data type schema, identified by the namespace-prefix *qdt1*.

After the BBIEs of an ABIE are evaluated, the XML generator processes the ASBIEs emanating from this ABIE. Line 12 - 15 show the definitions for the four ASBIEs emanating from *HoardingPermit*. Whereas the name of an BBIE is simply determined by taking the name specified by the attribute in the UML class diagram, ASBIEs are treated differently. The name of an ASIBE is determined by the role name of the ASBIE aggregation plus the name of the target ABIE, the ASBIE aggregation points to. The compound names of the four ASBIEs of *HoardingPermit* are *IncludedAttachement*, *CurrentApplication*, *IncludedRegistration* and *BillingPerson_Identification*. Multiplicities for ASBIEs are taken from the aggregation definition in the UML model. The type of an ASBIE is determined by the type of the ABIE, the aggregation points to. In the lines 12 - 14 the types are defined in a schema identified by the prefix *commonAggregates* which has been generated from the *BIELibrary CommonAggregates* shown in package 2 of figure 4.

Generating from a BIELibrary

The generation of a schema from a *BIELibrary* follows the same principle as the generation of a *DOCLibrary* schema. A *BIELibrary* itself can contain ABIEs from other *BIELibraries*, hence schemas of other *BIELibraries* are imported into the generated schema if necessary. In package 2 of figure 4 the *BIELibrary CommonAggregates* is shown. The ABIE *Person_Identification* has two ASBIEs emanating from it namely *Signature* and *Address*. One particularity is, that *Address* is connected by an *aggregation* while all the ABIE connections we saw so far were *compositions*. If an ASBIE is connected by a composition the ASBIE is first declared globally and then referenced in the ABIE it belongs to. Figure 7 shows a part of the schema generated for the *BIELibrary CommonAggregates*. In line 21 the ASBIE *AssignedAddress* is first declared globally and then referenced in line 26. This mechanism can also be applied in *DOCLibraries*.

```
20 [...]
21    <xsd:element name="AssignedAddress" type="commonAggregates:AddressType"/>
22    <xsd:complexType name="Person_IdentificationType">
23       <xsd:sequence>
24          <xsd:element name="Designation" type="udt2:IdentifierType"/>
25          <xsd:element name="PersonalSignature" type="commonAggregates:SignatureType"/>
26          <xsd:element ref="commonAggregates:AssignedAddress"/>
27       </xsd:sequence>
28    </xsd:complexType>
29 [...]
```

Figure 7. Compositions - defining the ASBIE globally

Generating from a CDTLibrary

Figure 8 shows a fraction of the schema, created for the

```
1  <xsd:schema xmlns:doc="urn:au:gov:vic:easybiz:data:draft:EB005-HoardingPermit" xmlns:commonAggregates="
   urn:au:gov:vic:easybiz:data:draft:CommonAggregates" xmlns:qdt1="urn:au:gov:vic:easybiz:types:draft:QualifiedDataTypes"
   xmlns:ccts="urn:un:unece:uncefact:documentation:standard:CoreComponentsTechnicalSpecification:2" xmlns:bie2="
   urn:au:gov:vic:easybiz:data:draft:LocalLawAggregates" xmlns:cdt1="un:unece:uncefact:data:standard:CDTLibrary:1.0"
   attributeFormDefault="unqualified" elementFormDefault="qualified" targetNamespace="
   urn:au:gov:vic:easybiz:data:draft:EB005-HoardingPermit" version="0.2" xmlns:xsd="http://www.w3.org/2001/XMLSchema">
2    <xsd:import schemaLocation="../urn_au_gov_vic_easybiz_/types_draft_coredatatypes_1.0.xsd" namespace="
   un:unece:uncefact:data:standard:CDTLibrary:1.0" />
3    <xsd:import schemaLocation="../urn_au_gov_vic_easybiz_/types_draft_BuildingAndPlanningDataTypes_0.1.xsd"
   namespace="urn:au:gov:vic:easybiz:types:draft:QualifiedDataTypes" />
4    <xsd:import schemaLocation="../urn_au_gov_vic_easybiz_/data_draft_CommonAggregates_0.1.xsd" namespace="
   urn:au:gov:vic:easybiz:data:draft:CommonAggregates" />
5    <xsd:import schemaLocation="../urn_au_gov_vic_easybiz_/data_draft_LocalLawAggregates_0.1.xsd" namespace="
   urn:au:gov:vic:easybiz:data:draft:LocalLawAggregates" />
6    <xsd:complexType name="HoardingPermitType">
7      <xsd:sequence>
8        <xsd:element minOccurs="0" name="ClosureReason" type="cdt1:TextType" />
9        <xsd:element minOccurs="0" name="IsClosedFootpath" type="cdt1:Indicator_CodeType" />
10       <xsd:element minOccurs="0" name="IsClosedRoad" type="qdt1:Indicator_CodeType" />
11       <xsd:element minOccurs="0" name="SafetyPrecaution" type="cdt1:TextType" />
12       <xsd:element minOccurs="0" maxOccurs="unbounded" name="IncludedAttachment" type="
   commonAggregates:AttachmentType" />
13       <xsd:element minOccurs="0" name="CurrentApplication" type="commonAggregates:ApplicationType" />
14       <xsd:element name="IncludedRegistration" type="bie2:RegistrationType" />
15       <xsd:element minOccurs="0" name="BillingPerson_Identification" type="commonAggregates:Person_IdentificationType"
   />
16     </xsd:sequence>
17   </xsd:complexType>
18   <xsd:element name="HoardingPermit" type="doc:HoardingPermitType" />
19 </xsd:schema>
```

Figure 6. XSD schema for the HoardingPermit DOCLibrary

CDTLibrary depicted in package 4 of figure 4. The fraction shows the type definition for the CDT *Code*. Similar to an ABIE, a core data type is defined as *complexType* in XML. However, it does not contain a *sequence* of elements but a *simpleContent* element whose extension base is the data type specified in the content component of the core data type. In our case the data type is *String* and hence

```
30 [...]
31     <xsd:complexType name="CodeType">
32       <xsd:simpleContent>
33         <xsd:extension base="xsd:string">
34           <xsd:attribute name="LanguageIdentifier" type="xsd:string" use="optional"/>
35           <xsd:attribute name="CodeListAgName" type="xsd:string" use="required"/>
36           <xsd:attribute name="CodeListName" type="xsd:string" use="required"/>
37           <xsd:attribute name="CodeListSchemeURI" type="xsd:string" use="required"/>
38         </xsd:extension>
39       </xsd:simpleContent>
40     </xsd:complexType>
41 [...]
```

Figure 8. XSD schema fraction for *CDTLibrary*

the build-in data type of the XML schema specification is taken. The supplementary components are defined as attributes of the *complexType*. The data type of an attribute and its multiplicity is again retrieved from the definition in the UML model.

Generating from a QDTLibrary

A schema generated from a *QDTLibrary* looks very similar to a schema generated from a *CDTLibrary*. Again, the data type specified in the content component determines the base for the extension. If an enumeration is used to restrict the possible values for the content component, the *complexType* of the enumeration is used for the restriction. In case the content component has no enumeration assigned to it, the *complexType* of the underlying core data type is used for the restriction.

Generating from ENUMLibray and PRIMLibrary

For every element stereotyped as ENUM in an *ENUMLibrary* a *simpleType* is created. The *simpleType* contains a restriction with base *xsd:token*. The values are then defined in *enumeration* tags. For PRIMLibraries currently no schema generation mechanism is implemented. Where primitive types are needed (*String*, *Integer* ...) the build-in types of the XSD schema are taken.

For the generation of the XML schemas, UML tagged values play an important role. Every library package within a business library has several tagged values, steering the generation process. The namespace of a specific schema for instance, is determined by the tagged value *baseURN*.

Apart from the namespace also the namespace prefix can be set by the user through tagged values. Line 12, 13 and 15 of figure 6 show the usage of a user specific prefix *commonAggregates*. The user specific namespace has been defined as a tagged value *NamespacePrefix* in the *BIELibrary CommonAggregates*. In case no user specific namespace prefix is set, a standard namespace prefix is taken. Line 14 shows the usage of a standard prefix for a *BIELibrary* namely *bie2*. The number contained in the prefix is generated automatically to distinguish between multiple *BIELibrary* schemas imported into a *DOCLibrary* schema.

As shown in figure 5 the user has the possibility to select whether the schema should be annotated or not. The

CCTS standard prescribes a set of annotations for every element of the standard. An ABIE for instance, amongst others, has two mandatory annotation fields *Version* and *Definition*. The solution approach taken by our XML generator is again driven by tagged values. The values for the different annotation fields are specified in tagged values. During a generation run the values are retrieved and transferred into the correct element annotations in the XML schema. For the sake of brevity annotations have been omitted in the sample schema, shown in figure 6.

5 Conclusion and Outlook

In this paper we have shown the implementation of a UML profile for core components and the generation of an XML schema from a core components model. The current development is a first step towards a tool supported modeling of core components and the automated generation of document artifacts. However, the current version provides only the basic generation and validation features - future release will gradually implement all features. In order to enhance the widespread use of core components, the potential user must be given a possibility to validate the created model. Even experienced core component modelers often get lost in a model because the interdependencies between CDTs, QDTs etc. blur with the increasing complexity of a model. Current effort is therefore spent on a validation engine, allowing to check the syntactical and semantical correctness of a core component model. Currently not all OCL constraints of the standard are evaluated. A future validation module will incorporate the complete OCL constraints specified in the standard [14] and be able to check the validity of the core components model accordingly. A valid model is also a crucial prerequisite for a model transformation as performed by the XML transformer. At the moment the transformer performs a basic model validation allowing to track and report basic flaws in the model. What is needed however, is a fine-grained model validation. Apart from the top priority extension of a model validator, several other features are also planned. Enterprise Architect only offers a very basic functionality for generating, moving and updating existing class definitions. The Add-In will therefore be extended by a core components management console, allowing the easy maintenance of existing libraries. Other modeler amenities such as updating all namespaces, setting one global schema location etc. are also subject to current development. Apart from tool driven changes we will also have to consider the changes which are made necessary by the release of the upcoming CCTS 3.0 standard.

After the implementations for core component modeling reach their final stage, the user will be given a powerful tool to model the information exchanged in a B2B business process.

References

[1] M. Bernauer, G. Kappel, and G. Kramler. Representing XML Schema in UML - A Comparison of Approaches. In *Proc. of 4th International Conference on Web Engineering, ICWE 2004*, pages 440–444. Springer LNCS 3140, 2004.

[2] R. Conrad, D. Scheffner, and J. C. Freytag. XML Conceptual Modeling Using UML. In *Proc. of the 19th Int'l Conf. on Conceptual Modeling, ER 2000*, pages 558–571. Springer LNCS 1920, 2000.

[3] E. Domínguez, J. Lloret, A. L. Rubio, and M. A. Zapata. Evolving XML Schemas and Documents Using UML Class Diagrams. In *16th Int'l Conf. on Database and Expert Systems Applications, DEXA 2005*, pages 343–352. Springer LNCS 3588, 2005.

[4] T. Kudrass and T. Krumbein. Rule-Based Generation of XML DTDs from UML Class Diagrams. In *7th East European Conference Advances in Databases and Information Systems, ADBIS 2003*, pages 339–354. Springer LNCS 2798, 2003.

[5] H. Li. XML and Industrial Standards for Electronic Commerce. *Knowledge and Information Systems*, 2(4):487–497, 2000.

[6] Y. Li and A. An. Representing UML Snowflake Diagram from Integrating XML Data Using XML Schema. In *Data Engineering Issues in E-Commerce*, pages 103–111. IEEE, 2005.

[7] P. Liegl, R. Schuster, and M. Zapletal. *UMM Add-In*. University of Vienna, 2006. Version 0.8.0, http://ummaddin.researchstudio.at.

[8] OASIS. *RELAX NG Specification*, December 2001. Committee Specification.

[9] OASIS, UN/CEFACT. *ebXML - Technical Architecture Specification*, February 2001. Version 1.4.

[10] N. Routledge, L. Bird, and A. Goodchild. UML and XML schema. In *ADC '02: Proceedings of the 13th Australasian database conference*, pages 157–166, Darlinghurst, Australia, Australia, 2002. Australian Computer Society, Inc.

[11] UN/CEFACT. *Electronic Data Interchange*, September 2006. D.06A.

[12] UN/CEFACT Applied Technology Group (ATG). *XML Naming and Design Rules*, February 2006. 2.0.

[13] UN/CEFACT TMG. *Core Components Technical Specification - Part 8 of the ebXML Framework*, November 2003. v2.01.

[14] UN/CEFACT TMG. *BCSS - UML Profile for Core Components based on CCTS 2.01*, October 2006. Candidate for version 1.0.

[15] W3C. *RDF Vocabulary Description Language*, February 2004. W3C Recommendation 1.0.

[16] World Wide Web Consortium (W3C). *XML Schema*, 2004. Version 1.0.