Binding UMM Business Documents to a Business Document Ontology

Birgit Hofreiter
Faculty of Computer Science, University of Vienna, Liebiggasse 4, 1010 Vienna, Austria
University of Technology Sydney, PO Box 123 Broadway, NSW 2007, Australia
birgit.hofreiter@univie.ac.at

Abstract

UN/CEFACT's modeling methodology (UMM) is used to develop business collaboration models independent of the IT-platform. The information being exchanged is modeled by class diagrams. It is expected that later on these class diagrams are mapped to business document standards. Since it is hard to predict which standards will be dominant in the future, we do not propose direct mappings from UMM business information class diagrams to the specific business document languages. Instead we propose an ontology layer to represent the information identified in UMM. This ontology layer is based on UN/CEFACT'score components. For this purpose we have developed an RDFS schema (RDFS) for the core components meta model. Furthermore, we define a mapping from the UMM class diagram to the core component RDFS.

1 Motivation

The development of any good system of a certain size requires a well defined analysis and design process. For the specific focus of developing B2B systems, the United Nation's Centre for Trade Facilitation and Electronic Business (UN/CEFACT) has developed the UN/CEFACT modeling methodology (UMM) for analyzing both dynamic and structural B2B aspects [UN06]. A UMM model concentrates on the business aspects only. It is not bound to a specific platform. In order to implement a B2B system a UMM model must be transformed to platform specific specifications. This means the dynamic aspects will be transformed to a business process modelling language (e.g. BPEL, BPSS). We describe these mappings in [HoHu04] and [HoHK06]. The structural aspects will be mapped to business document standards which is a focus of this paper.

This leads to the question which business document standard should be supported? Although most implementations run on UN/EDIFACT, all latest implementations are based on XML. Therefore, a mapping to an XML-based business document standard is preferred. Unfortunately, plenty of these business document standards exist. Although some of the business document standards have become the first choice within a vertical industry, there are still some competing efforts. Popular business document standards include those of market place providers, like Commerce One's xCBL and Ariba's cXML, the Open Application Group's OAGI interconnecting ERP systems, OASIS's UBL and domain-specific solutions like RosettaNet in the IT sector. An overview of business document standards is provided in [Li00].

After the e-commerce hype only a few new business document standards appeared and there is a trend that some standards merge or disappear. Today we are not in a stable situation and it is hard to forecast which will be the most commonly accepted business document standard in the long run. For example, five years ago xCBL was one of the most accepted business document standards. Today its support is rather limited.

Instead of developing a mapping from UMM class diagrams to a specific business document language, of which we do not know how it will look like tomorrow, we prefer an approach that will survive industry trends. Therefore, we propose a business document ontology layer that is first of all independent of the XML-schema of a business document language. This ontology layer is based on UN/CEFACT's core components. We use RDFS to describe the core components based ontology. Furthermore we use RDF's XML serialization to denote the ontology in a machine-readable format. This is a precondition that the mapping from UMM class diagrams to our business document ontology is useful for a further transformation to a business document language of choice.

The remainder of this paper is structured as follows: In section 2 we show how the core components-based business document ontology works. By the example of a quote in our purchase order management case study, we demonstrate a mapping from a UMM class diagram to our business document ontology in section 3. In section 4 we define what is required for a mapping from the business document ontology to a business document language.

2 Related Work

The rapid growth in information volume makes it more and more difficult to find, organize, access and maintain the information required by users. The Semantic Web proposes enhanced information access based on machine-processable meta-data. This resulted in an increased interest of computer scientists in ontologies. However, the term ontology is used for a much longer time in philosophy. There ontology means the study of being or existence as well as the basic categories thereof. In computer science, an ontology is the attempt to formulate an exhaustive and rigorous conceptual schema within a given domain, a typically hierarchical data structure containing all the relevant entities and their relationships and rules within that domain. Ontologies are commonly used in artificial intelligence and knowledge representation. T.R. Gruber has described an ontology as *"a formal, explicit specification of a shared conceptualization"* [Grub93]. A conceptualization is an abstract, simplified view of the world that we wish to represent. It should be shareable and re-usable.

An ontology is always developed for a certain purpose. As a result, ontologies developed independently for different purposes will differ significantly from each other. An ontology which is subject-independent is known as a *foundation ontology* or *upper ontology.* An upper ontology is limited to concepts that are meta, generic, abstract, and hence are general enough to address (at a high level) a broad range of domain areas.

No upper ontology has yet become a de-facto standard. Different organizations are attempting to define standards for specific domains. For practical reasons, ontologies must be expressed in a concrete notation. An ontology language is a formal syntax by which an ontology is built. Over the last years a number of ontology languages have been developed, both proprietary and standards-based. The most well-known ontology languages are the following:

The Resource Description Framework (RDF) is a World Wide Web Consortium (W3C) standard recommendation for describing machine-processable semantics of data represented by subject-property-object triples [MaMi04]. RDFS is also a W3C recommendation and provides a means to define the vocabulary for RDF properties and specifies the kinds of objects to which these properties may be applied. This means, RDFS provides a basic type system for RDF data models [BrGu02]. DARPA Agent Markup Language (DAML) was created as part of a research program started in August 2000 by DARPA, a US governmental research organization; and Ontology Inference Layer (OIL) is an initiative funded by the European Union programme for Information Society Technologies as part of some of its research projects. DAML and OIL are standardized schema languages and not data models like RDF or topic maps [W3C01]. The Web Ontology Language (OWL) has been developed by the Web Ontology Working Group as part of the W3C Semantic Web Activity and has the status of a recommendation since February 2004 [McGH04]. It is designed for use by applications that need to process the content of information instead of just presenting information. The working group proposes that OWL facilitates greater machine interpretability of Web content than that supported by RDF and RDFS. XML Topic Maps (XTM) are a reformulation of topic maps in XML syntax based on XLink. Topic maps have a long and complicated history. They originally started as a process to create a standard SGML DTD for software documentation.

3 A Core Components Based Ontology

The goal of our B2B business document ontology is to describe the semantic concepts to be represented in document types exchanged between business partners [HoHu02]. It is important that document types are not developed in isolation from each other. Semantic concepts that are shared between different document types must be re-used. This means, e.g., that a line item should be based on the same concept regardless if it is a quote document type from one or the other business environment, or a purchase order, an invoice, etc.

In order to develop a business document ontology, two main approaches are introduced by Ontoprise's Semantic B2B Broker [Onto00]: a top-down (Fig. 1a) and a bottom-up approach (Fig. 1b). A top-down approach is used if no XML structure of a business document standard is given and such a structure is secondary. In this case, business experts will first define a document ontology that describes their shared understanding of a given business document type. The conceptual model of this document ontology builds the foundation of the development of a new business document standard (represented as DTD or XML schema). Vice versa, a bottom-up approach takes DTDs or XML schemas from existing business document standards to analyze their semantic content. The conceptual models of all considered business document standards have to be harmonized in order to define a unified document ontology. A bottom-up approach is used in [OmFe00] to mediate between Commerce One's xCBL and Ariba's cXML.

The industry-wide recognized development of components independent of a business document standard is currently undertaken by UN/CEFACT. The work started as part of the ebXML initiative and is called UN/CEFACT's core components. It is our goal to take advantage of a future pool of core components. Hence, our ontology layer is not defined by reverse engineering of existing business document standards like in the bottom-up approach. Instead, our ontology layer is based on UN/CEFACT's core components. However, we do not use a pure top-down approach, because we will not develop a new business document standard. We have to mediate the core components-based ontology layer with existing business document standards. Since we are coming from top as well as from bot-

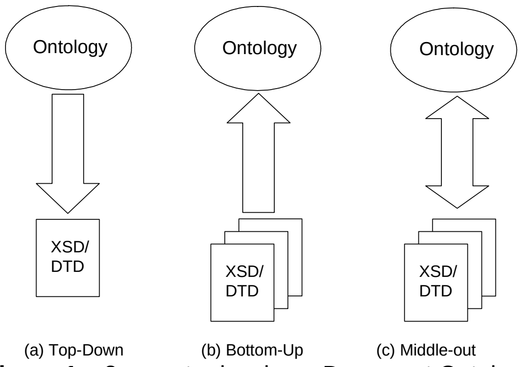

Figure 1. 3 ways to develop a Document Ontology

tom, we call our approach “middle-out”. This approach is displayed in Fig. 1c.

In the first step we develop an ontology that follows the UN/CEFACT’s core components specification [UN03]. For this purpose we have developed an RDF schema (RDFS) [BrGu02] for the core components meta model. The resulting RDFS is depicted as a graph in Fig. 2 and listed in the code line 1 to line 52 further below. All boxes with solid lines are of the RDFS type *Class* and those with broken lines are of type *Property*. It is important to note that each defined class is also defined as a subtype of *Class* and, thus, inherits the concepts of *Class*. This trick allows that an instance of this schema, is by itself a schema for describing business document types.

A *core component* is defined as a semantic concept that is shared between different types of business documents. If a *core component* is used in a business document type, it is set into the business context where the document type is used. Usually, it is adjusted to the needs of the business context. This happens when a *core component* becomes a *business information entity.* We use the abstract concept of a *building block* as a super-class for *core component* and *business information entity.*

A *core component* is defined as a semantic concept that is shared between different types of business documents. If a *core component* is used in a business document type, it is set into the business context where the document type is used. Usually, it is adjusted to the needs of the business context. This happens when a *core component* becomes a *business information entity.* We use the abstract concept of a *building block* as a super-class for *core component* and *business information entity.*

There exist 3 different types of core components. A *basic core component* represents a singular business concept with a unique semantic definition. Each basic core component is of a certain core component type. A *core component type* (e.g. amount type) consists of a *content component* that carries the actual content (e.g. 12) plus one or more *supplementary components* giving an essential extra definition to the content component (e.g. Euros). Note, that content and supplementary components are nothing else than core components. Core component types do not have business meaning. An *aggregate core component* comprises other core components that convey a distinct business meaning together.

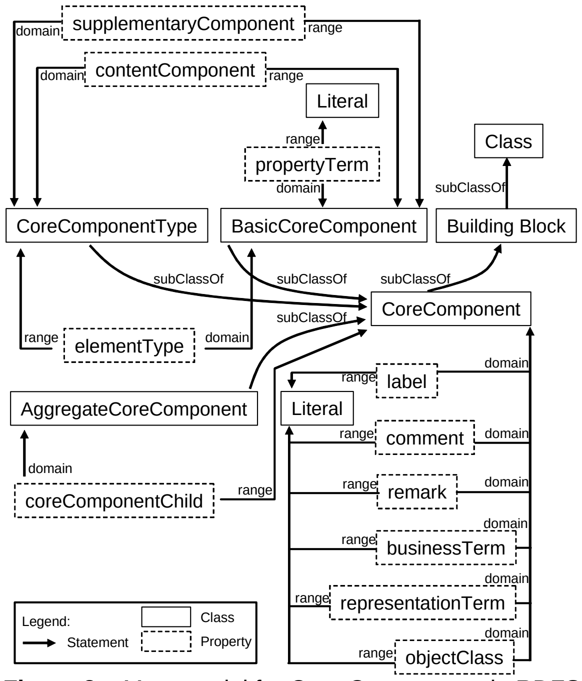

Figure 2. Meta model for Core Components in RDFS

In the RDF schema the three different types *aggregate core component* (line 24), *basic core component* (line 31), and *core component type* (line 42) are represented as subclasses of the class *core component* (line 4). The property *element type* (line 38) is used to assign a core component type to a basic core component. The composition of a core component type is defined by the properties *content component* (line 45) and *supplementary component* (line 49), referencing basic core components. The property *core component child* (line 27) is used to reference the components within an aggregate.

```
[1]     <rdfs:Class rdf:ID="BuildingBlock">
[2]         <rdfs:subClassOf rdf:resource="http://w3.../rdf-schema#Class"/>
[3]     </rdfs:Class>
[4]     <rdfs:Class rdf:ID="CoreComponent">
[5]         <rdfs:subClassOf rdf:resource="#BuildingBlock"/>
[6]         <rdfs:comment>Inherits theLabel and Comment from class Class
            </rdfs:comment>
[7]     </rdfs:Class>
[8]     <rdf:Property ID="remark">
[9]         <rdfs:domain rdf:resource="#CoreComponent"/>
[10]        <rdfs:range rdf:resource="http://w3.../rdf-schema#Literal"/>
[11]    </rdf:Property>
[12]    <rdf:Property ID="businessTerm">
[13]        <rdfs:domain rdf:resource="#CoreComponent"/>
[14]        <rdfs:range rdf:resource="http://w3.../rdf-schema#Literal"/>
[15]    </rdf:Property>
[16]    <rdf:Property ID="representationTerm">
[17]        <rdfs:domain rdf:resource="#CoreComponent"/>
[18]        <rdfs:range rdf:resource="http://w3.../rdf-schema#Literal"/>
[19]    </rdf:Property>
[20]    <rdf:Property ID="objectClass">
[21]        <rdfs:domain rdf:resource="#CoreComponent"/>
[22]        <rdfs:range rdf:resource="http://w3.../rdf-schema#Literal"/>
[23]    </rdf:Property>
[24]    <rdfs:Class rdf:ID="AggregateCoreComponent">
[25]        <rdfs:subClassOf rdf:resource="#CoreComponent"/>
[26]    </rdfs:Class>
[27]    <rdf:Property ID="coreComponentChild">
[28]        <rdfs:domain rdf:resource="#AggregateCoreComponent"/>
[29]        <rdfs:range rdf:resource="#CoreComponent"/>
[30]    </rdf:Property>
[31]    <rdfs:Class rdf:ID="BasicCoreComponent">
[32]        <rdfs:subClassOf rdf:resource="#CoreComponent"/>
[33]    </rdfs:Class>
[34]    <rdf:Property ID="propertyTerm">
[35]        <rdfs:domain rdf:resource="#BasicCoreComponent"/>
[36]        <rdfs:range rdf:resource="http://w3.../rdf-schema#Literal"/>
[37]    </rdf:Property>
[38]    <rdf:Property ID="elementType">
[39]        <rdfs:domain rdf:resource="#BasicCoreComponent"/>
[40]        <rdfs:range rdf:resource="#CoreComponentType"/>
[41]    </rdf:Property>
[42]    <rdfs:Class rdf:ID="CoreComponentType">
[43]        <rdfs:subClassOf rdf:resource="#CoreComponent"/>
[44]    </rdfs:Class>
[45]    <rdf:Property ID="contentComponent">
[46]        <rdfs:domain rdf:resource="#CoreComponentType"/>
[47]        <rdfs:range rdf:resource="#BasicCoreComponent"/>
[48]    </rdf:Property>
```

```
[49]   <rdf:Property ID="supplementaryComponent">
[50]       <rdfs:domain rdf:resource="#CoreComponentType"/>
[51]       <rdfs:range rdf:resource="#BasicCoreComponent"/>
[52]   </rdf:Property>
```

Each UN/CEFACT core component contains the following dictionary information: A dictionary entry name is the unique official name of the core component. It corresponds to the RDFS property *label*. The definition of the unique semantic business meaning of the core component is given in the RDFS property *comment*. Therefore, we do not have to redefine these two properties in our code. The property *remark* (line 8) is used to further clarify the definition, to provide examples and/or to reference a recognized standard. If there exist further synonym terms under which the core component is commonly known and used in the business, the property *business term* (line 12) is used to define them. For reasons of completeness, we assign the properties *representation term (*line 16), *object class* (line 20) to core components, and *property term* (line 34) only to basic core components as defined in the UN/CEFACT specification.

The RDF library of core components must be populated with all core components currently being developed by UN/CEFACT. Each core component will be expressed as an RDF model that follows the RDFS of the core components' meta model. The code line 53 to line 108 demonstrates the population of the RDF library by the means of the aggregate core component *product service details*. A representation of this core component as UML class could be represented as follows:

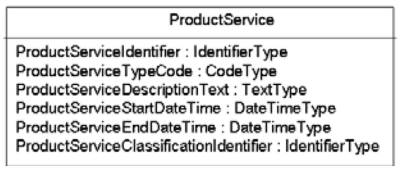
ProductService
ProductServiceIdentifier : IdentifierType
ProductServiceTypeCode : CodeType
ProductServiceDescriptionText : TextType
ProductServiceStartDateTime : DateTimeType
ProductServiceEndDateTime : DateTimeType
ProductServiceClassificationIdentifier : IdentifierType

Figure 3. UML Class for Product Service

The resource representing this aggregate core component (line 97) is marked *#000155* according to its UN/CEFACT core component ID. This represents only the fragment identifier of a URI uniquely identifying core components. The unique identifications (UID) assigned to core components are used as fragment identifier. The label of the resource is equal to the data dictionary entry name *product service details* (line 98). The comment states the definition of the *product service details* (line 99). The business term *good* which is commonly used to refer to *product service* is assigned to it (line 100). Each of the components aggregated within *product service details* is assigned as *core component child* (line 102 to line 107).

As an example, we have a detailed look on the component *product service identifier.* This basic core component is defined in line 65 to line 71. Its identifier is #000156 (line 65). It is referenced in the *coreComponentChild* statement of the aggregate core component *service product details* (line 102*)*. Similarly to aggregate core components, the *product service identifier* captures label, comment and objectClass (line 66 to line 68). Since it is a basic core component it also carries a propertyTerm (line 69) and is assigned to a specific type (line 70). Since product service identifier is of type identifier the elementType statement in line 70 references the ID #000101 of the core component type *identifier type* (line 62*)*. In the example code we just show the label of the core components type (line 63) and ommit to present all its other characteristics.

All the other basic core components that are children of the product service details (and correspond to the attributes in Fig. 3) are rudimentarily outlined in line 72 to line 96. All their data types are defined in line 53 to line 64.

```
[53]    <cc:CoreComponentType rdf:ID="000066">
[54]        <rdfs:label xml:lang="en">DateTime.Type</rdfs:label>
[55]    </cc:CoreComponentType>
[56]    <cc:CoreComponentType rdf:ID="000089">
[57]        <rdfs:label xml:lang="en">Code.Type</rdfs:label>
[58]    </cc:CoreComponentType>
[59]    <cc:CoreComponentType rdf:ID="000090">
[60]        <rdfs:label xml:lang="en">Text.Type</rdfs:label>
[61]    </cc:CoreComponentType>
[62]    <cc:CoreComponentType rdf:ID="000101">
[63]        <rdfs:label xml:lang="en">Identifier.Type</rdfs:label>
[64]    </cc:CoreComponentType>
[65]    <cc:BasicCoreComponent rdf:ID="000156">
[66]        <rdfs:label xml:lang="en">ProductService.Identifier</rdfs:label>
[67]        <rdfs:comment>A character string used to uniquely identify and
            distinguish a product/service.</rdfs:comment>
[68]        <cc:objectClass>ProductService</cc:objectClass>
[69]        <cc:propertyTerm>Identification</cc:propertyTerm>
[70]        <cc:elementType cc:CoreComponentType="#000101"/>
[71]    </cc:BasicCoreComponent>
[72]    <cc:BasicCoreComponent rdf:ID="000157">
[73]        <rdfs:label xml:lang="en">ProductService.Type.Code</rdfs:label>
[74]        ...
[75]        <cc:elementType cc:CoreComponentType="#000089"/>
[76]    </cc:BasicCoreComponent>
[77]    <cc:BasicCoreComponent rdf:ID="000158">
[78]        <rdfs:label...>ProductService.Description.Text</rdfs:label>
[79]        ...
[80]        <cc:elementType cc:CoreComponentType="#000090"/>
[81]    </cc:BasicCoreComponent>
[82]    <cc:BasicCoreComponent rdf:ID="000159">
[83]        <rdfs:label ...>ProductService.Start.DateTime</rdfs:label>
[84]        ...
[85]        <cc:elementType cc:CoreComponentType="#000066"/>
[86]    </cc:BasicCoreComponent>
[87]    <cc:BasicCoreComponent rdf:ID="000160">
[88]        <rdfs:label ...>ProductService.End.DateTime</rdfs:label>
[89]        ...
[90]        <cc:elementType cc:CoreComponentType="#000066"/>
[91]    </cc:BasicCoreComponent>
[92]    <cc:BasicCoreComponent rdf:ID="000163">
[93]        <rdfs:label ...>ProductServiceClassification.Identifier</rdfs:label>
[94]        ...
[95]        <cc:elementType cc:CoreComponentType="#000101"/>
[96]    </cc:BasicCoreComponent>
[97]    <cc:AggregateCoreComponent rdf:ID="000155">
[98]        <rdfs:label xml:lang="en">ProductService.Details</rdfs:label>
[99]        <rdfs:comment>A thing or substance produced by natural process
            or manufacturer and or a provision or system of supplying a need.
            </rdfs:comment>
[100]       <cc:businessTerm>Good</cc:businessTerm>
[101]       <cc:objectClass>ProductService</cc:objectClass>
[102]       <cc:coreComponentChild cc:CoreComponent="#000156"/>
[103]       <cc:coreComponentChild cc:CoreComponent="#000157"/>
[104]       <cc:coreComponentChild cc:CoreComponent="#000158"/>
[105]       <cc:coreComponentChild cc:CoreComponent="#000159"/>
[106]       <cc:coreComponentChild cc:CoreComponent="#000160"/>
[107]       <cc:coreComponentChild cc:CoreComponent="#000163"/>
[108]   </cc:AggregateCoreComponent>
```

Besides *product service details* all the other core components must be described by RDFS and provided in a library. Currently, UN/CEFACT is in the process of delivering a first full set of core components. In the future, these are the ones that must be available for real world use. For demonstration purposes we use the preliminary set of core components as issued by UN/CEFACT.

4 From a UMM Class Diagram to the Business Document Ontology

The core components stored in a core components library are the foundation for building business document types. In order to develop a business document type the question is which core components are used and how are they assembled? In order to identify the core components it is a good idea to start with UMM. The UMM model defines the business information being exchanged during a business transaction. The business process oriented approach guarantees that the business information is kept to the minimum necessary. The resulting business document semantics are modeled in a class diagram. This class diagram is already built from classes that correspond to core component semantics. Therefore, classes and their attributes give a hint to those core components that must be selected from the library in order to build the corresponding business document type.

Core components are by definition free of a business context. Business document types are always used in a certain business context. Accordingly, a building block actually used in a business document type is not called a core component anymore. This building block is now called business information entity. A business information entity adopts a core component to the business context of its usage and, thus, is always based on a core component. A business document type is always an assembly of business information entities. This means that it is necessary to describe the relationships of the business information entities within the assembly. Therefore, it is necessary to define how business information entities are associated with other ones.

In order to meet these requirements we have to extend our meta model of Fig. 2. The necessary extensions are depicted in Fig. 4 and are listed in code line 109 to line 135. A *business information entity* is defined as a subclass of the abstract concept *building block* (line 109). Accordingly, *building block* is the abstract super class for both *core component* and *business information entity.* A business information entity is based on a core component. However, it is not necessary that it is directly based on a core component. This means that a business information entity might be based on another business information entity. Hence, the property *based on* points to the abstract *building block* a business information entity is based on (line 112). This allows to point to both a *core component* and a *business information entity.*

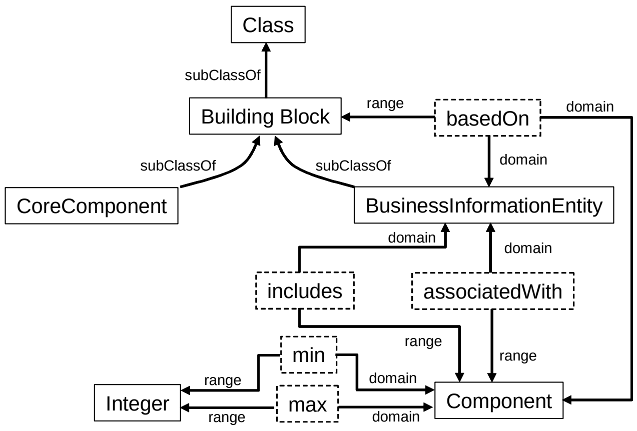

Figure 4. Meta Model for Business Information Entities

The property *associated with* is used for referencing the business information entity to which an association exists. Nevertheless, the property does not directly point to another *business information entity.* For a business information entity instance it is important to define how many instances of the related business information entity are associated with it. RDFS by itself does not support the specification of lower and upper limits for statements of a certain type. To overcome this limitation we define the class *component* (line 121). The property *associated with* points to a *component* (line 117). Component carries the *min* (line 124) and the *max* (line 128) property to define the lower and upper bound of the related business information entity. Of course the *component* must point to the business information entity that is related to the business information entity in question. We define on which *building block* a *component* is based upon. For this purpose, we use the property *based on* (line 112) again. Thus, the domain of *based on* is not only *business information entity* (line 113), but also *component* (line 114).

The next constraint we have to consider is that not all of the attributes of a class are relevant in a certain business context. This type of constraint leads to the fact that a business information entity is not used exactly as the underlying core component. For this purpose we define the property *includes* (line 132) that is used to define the children of an *business information entity.* Similarly to the *associated with* property, the property *includes* does not directly reference the included core components or business information entities. It points to a *component* which in turn references the included *building block* (*core component* or *business information entity*). We have already explained that a component specifies the *min*imum and *max*imum occurrence of the included building block. A *max*imum value of 0 signals that the attribute is not used.

```
[109]  <rdfs:Class rdf:ID="BusinessInformationEntity">
[110]      <rdfs:subClassOf rdf:resource="http://w3.../rdf-schema#Class"/>
[111]  </rdfs:Class>
[112]  <rdf:Property rdf:ID="basedOn">
[113]      <rdfs:domain rdf:resource="#BusinessInformationEntity"/>
[114]      <rdfs:domain rdf:resource="#Component"/>
[115]      <rdfs:range rdf:resource="#BuildingBlock"/>
[116]  </rdf:Property>
```

```
[117]  <rdf:Property rdf:ID="associatedWith">
[118]      <rdfs:domain rdf:resource="#BusinessInformationEntity"/>
[119]      <rdfs:range rdf:resource="#Component"/>
[120]  </rdf:Property>
[121]  <rdfs:Class rdf:ID="Component">
[122]      <rdfs:subClassOf rdf:resource="Class"/>
[123]  </rdfs:Class>
[124]  <rdf:Property rdf:ID="min">
[125]      <rdfs:domain rdf:resource="#AggregateBusinessInformationEntity"/>
[126]      <rdfs:range rdf:resource="#Integer"/>
[127]  </rdf:Property>
[128]  <rdf:Property rdf:ID="max">
[129]      <rdfs:domain rdf:resource="#AggregateBusinessInformationEntity"/>
[130]      <rdfs:range rdf:resource="#Integer"/>
[131]  </rdf:Property>
[132]  <rdf:Property rdf:ID="includes">
[133]      <rdfs:domain rdf:resource="BusinessInformationEntity"/>
[134]      <rdfs:range rdf:resource="#Component"/>
[135]  </rdf:Property>
```

The model developed so far allows us to assemble business document types from existing core components in a library as well as to create business information entities that customize the underlying core components to the specific needs of a certain business context.

In order to demonstrate our approach we take the example of a *quote* document. We assume that a business process analyst has followed the UMM and identified the information depicted in the class diagram of Fig. 5 to be interchanged during a business collaboration. This class diagram is reduced to show only those elements we need to demonstrate our approach and, consequently, does not show attributes for most of the classes. As mandated by the UMM this class diagram is already based on classes that are equivalent to core components. A quick look on Fig. 5 tells us, that we do not only need the information concerning product service details, but also core components for unit charge price (#000125), line (#000135), and document details (#000210) - the latter being the base for *quote* and *quote request*. For the transformation process we suppose that all the necessary core components are stored in the core components library.

As we said before, as soon as a core component is used in a business document type it becomes a business information entity. Therefore, we first have to select the core components we need from the library. For each core component we create (at least) one business information entity. A quote is a special type of document. Hence, we select the core component *#000210 document details* from the library

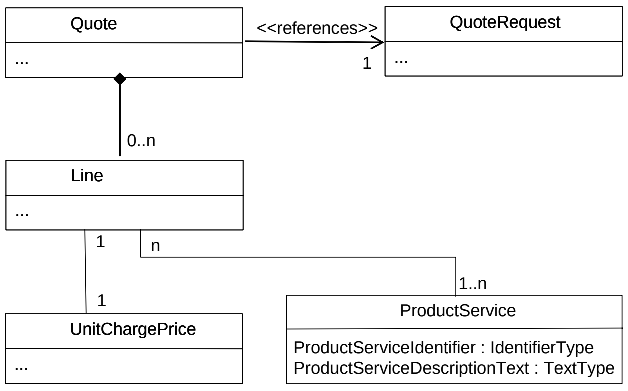

Figure 5. Class Diagram for a Context-Free Quote

and create a business information entity *quote* that is based on *#000210* (line 136). Similarly, we create a business information entity *quote request* that is based on the same core component *#000210 document details* (line 141). We base the business information entity *line* on the core component #000135 *line identifier* (line 144). Furthermore, we create a business information entity *unit charge price* from the core component #000125 *unit charge price details* (line 149) and a business information entity *product/service* from the core component #000155 *product/service details* (line 152). Each of the business information entities used in the business document has exactly the same structure as the underlying core component defined in the library.

Having created the five business information entities *quote*, *quote request*, *line*, *unit charge price* and *product/ service*, we have to establish the necessary associations between them. A quote is made as a response to exactly one quote request. In the terminology of our business document ontology this means a *quote* is associated with a component - named *corresponding quote request* - that is based on the business information entity *quote request*. This component is specified exactly one time, since the *minimum* and the *maximum* occurrence is *1* (line 138, line 161 and line 165). Furthermore, the business information entity *quote* is associated with *0* to *n* components - called *quoted line* - that are based on the business information entity *line* (line 139, line 166 to line 170).

The business information entity *line* is associated with exactly *1* component *quoted price for line* that is based on *unit charge price* (line 146, line 171 to line 175). Finally, a *line* is associated with *1* to *n* components *products/service of line* that are based on the business information entity *product/service* (line 147, line 176 to line 180). It should be noted that it is up to the document designer whether an association is created for both directions or only one. In absence of names of associations and roles in the UMM class diagram, it is up to the document designer to give names to the *components* that are pointed to by the *associated with* property.

In our example the business information entity product service (line 152) is based on its corresponding core compoent (line 153, line 97). However, the business information entity uses only 2 out of the 6 children of the core component. Thus, the business information entity references a component for each of the 6 children (line 154 to line 159). Since our example uses only the *product/service identifier* and *product/service description text* the *min* and *max* property of these components are set to *1* (line 181 to line 185 and line 190 to line 194). Since the other four properties are not used in the book example, the corresponding *max* property is set to *0* (line 186 to line 189 and line 195 to line 206).

```
cc:BusinessInformationEntity rdf:ID="Quote">
    <cc:basedOn rdf:resource="http://my.../cc-libary#000210"/>
    <cc:associatedWith rdf:ID="#CorrespondingQuoteRequest"/>
    <cc:associatedWith rdf:ID="#QuotedLine"/>
</cc:BusinessInformationEntity>
<cc:BusinessInformationEntity rdf:ID="QuoteRequest">
    <cc:basedOn rdf:resource="http://my.../cc-libary#000210"/>
</cc:BusinessInformationEntity>
<cc:BusinessInformationEntity rdf:ID="Line">
    <cc:basedOn rdf:resource="http://my.../cc-libary#000135"/>
    <cc:associatedWith rdf:ID="#QuotedPriceForLine"/>
    <cc:associatedWith rdf:ID="#ProductServiceOfLine"/>
</cc:BusinessInformationEntity>
<cc:BusinessInformationEntity rdf:ID="UnitChargePrice">
    <cc:basedOn rdf:resource="http://my.../cc-libary#000125"/>
</cc:BusinessInformationEntity>
<cc:BusinessInformationEntity rdf:ID="ProductService">
    <cc:basedOn rdf:resource="http://my.../cc-libary#000155"/>
    <cc:includes rdf:ID="#ProductServiceIdentifier"/>
    <cc:includes rdf:ID="#ProductServiceTypeCode"/>
    <cc:includes rdf:ID="#ProductServiceDescriptionText"/>
    <cc:includes rdf:ID="#ProductServiceStartDateTime"/>
    <cc:includes rdf:ID="#ProductServiceEndDateTime"/>
    <cc:includes rdf:ID="#ProductServiceClassificationIdentifier"/>
</cc:BusinessInformationEntity>
<cc:Component rdf:resource="CorrespondingQuoteRequest">
    <cc:basedOn rdf:resouce="#QuoteRequest"/>
    <cc:min>1</cc:min>
    <cc:max>1</cc:max>
</cc:Component>
<cc:Component rdf:resource="QuotedLine">
    <cc:basedOn rdf:resouce="#Line"/>
    <cc:min>0</cc:min>
    <cc:max>unbounded</cc:max>
</cc:Component>
<cc:Component rdf:resource="QuotedPriceForLine">
    <cc:basedOn rdf:resouce="#UnitChargePrice"/>
    <cc:min>1</cc:min>
    <cc:max>1</cc:max>
</cc:Component>
<cc:Component rdf:resource="ProductServiceOfLine">
    <cc:basedOn rdf:resouce="#ProductService"/>
    <cc:min>1</cc:min>
    <cc:max>unbounded</cc:max>
</cc:Component>
<cc:Component ref:resource="ProductServiceIdentifier">
    <cc:basedOn ref:resource="http://my.../cc-libary#000156"/>
    <cc:min>1</cc:min>
    <cc:max>1</cc:max>
</cc:Component>
<cc:Component ref:resource="ProductServiceTypeCode">
    <cc:basedOn ref:resource="http://my.../cc-libary#000157"/>
        <cc:max>0</cc:max>
</cc:Component>
<cc:Component ref:resource="ProductServiceDescriptionText">
    <cc:basedOn ref:resource="http://my.../cc-libary#000158"/>
    <cc:min>1</cc:min>
    <cc:max>1</cc:max>
</cc:Component>
<cc:Component ref:resource="ProductServiceStartDateTime">
    <cc:basedOn ref:resource="http://my.../cc-libary#000159"/>
    <cc:max>0</cc:max>
</cc:Component>
<cc:Component ref:resource="ProductServiceEndDateTime">
    <cc:basedOn ref:resource="http://my.../cc-libary#000160"/>
    <cc:max>0</cc:max>
</cc:Component>
<cc:Component ref:resource="ProductServiceClassificationIdenti-
fier">
    <cc:basedOn ref:resource="http://my.../cc-libary#000163"/>
    <cc:max>0</cc:max>
</cc:Component>
```

5 Language Binding for Business Document Standards

Our business document ontology should be used in a next step to provide language bindings to corresponding document types in a business document language. Since our “middle-out” approach and a bottom-up approach only differ in the way the ontology is built, the problem of defining a language binding remains the same. Thus, our framework considers the language binding defined in the bottom-up approach in [OmFe00].

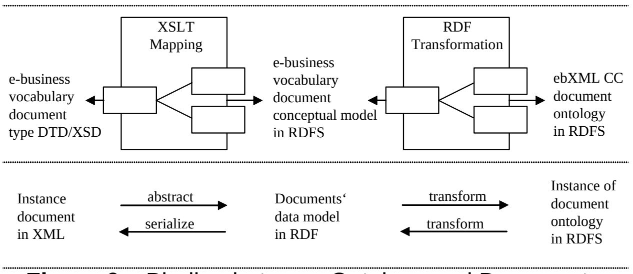
XSLT Mapping
RDF Transformation
e-business vocabulary document type DTD/XSD
e-business vocabulary document conceptual model in RDFS
ebXML CC document ontology in RDFS
Instance document in XML
abstract
serialize
Documents' data model in RDF
transform
transform
Instance of document ontology in RDFS

Figure 6. Binding between Ontology and Documents

The basic concept of this language binding is depicted in Fig. 6. It is based on the definition of a common conceptual model for document types of different business document standards. The conceptual models are described in RDFS. Thus, a mapping between the business document language’s DTD or XML schema on the one side and the conceptual data model on the other side is required. This mapping on the schema level is defined by the means of XSLT. On the instance level, an incoming document is abstracted from its XML serialization and translated into its RDF data model. Vice versa, in order to create an outgoing document, the RDF data model of the target business document language is serialized according to the target XML format.

Furthermore, the conceptual data model of a business document language’s document type (expressed in RDFS) must be mapped to the document ontologies data model of the same document type (also expressed in RDFS). This means that the equivalent mapping between the terminologies requires a transformation from one RDFS model to the other. The mapping must be described by the means of an RDFS mapping language.The mappings must be automatically translated into an RDF transformation language. This transformation language is applied to translate the conceptual RDF model of an incoming document into an instance of the document ontology and, vice versa, to translate an instance of the document ontology into the conceptual RDF model of an outgoing document. It follows that all mappings between different business document standards will also be managed by mapping each standard to the core components-based document ontology. Thus, future work has to concentrate on a RDFS mapping language and the RDF transformation language.

6 Conclusion

In this paper we presented an approach to define a business document ontology using RDFS. The business documents are built upon the UN/CEFACT core components concept. Core components are re-usable building blocks, which we represent by the means of RDFS. The RDFS notation of core components provides a basis for a mapping to different XML-based business document standards.

An evaluation of our approach according to the set of quality criterias for ontologies discussed in Gruber [Grub93] results in the conclusion that we meet every single criteria. *Clarity* requires definitions to be context independent. Core components are said to be free of context. In fact, they are not limited to any business sector. They are only assigned to the very general domain of „business“. *Coherence* demands consistency of the definitions. Core components are based on a controlled vocabulary in order to ensure coherence. *Extensibility* is reached if new concepts can easily be composed without any changes to the ontological foundations. The usefulness of an ontology depends on its extensibility. This is very important in dynamic environments like business. Our ontology proposed might be extended to meet the requirements of future business processes.

Another criteria is a *minimal encoding bias*. An encoding bias results when representation choices are made only for the convenience of notation or implementation. The development of our ontology was not influenced by any programming or encoding environment. Finally, Gruber asks for *minimal ontological committment*. An ontology should make as few claims as possible about the world being modeled, allowing the parties committed to the ontology freedom to specialize and instantiate the ontology as needed. In our proposed ontology parties will always have to specialize and use the ontology as needed in their business environment.

References

[BrGu02] *Brickley, D., Guha, R.:* RDF Vocabulary Description Language 1.0: RDF Schema, W3C Working Draft, 2002, http://www.w3.org/TR/2002/WD-rdf-schema-20020430

[Grub93] *Gruber, T.R.:* A Translation Approach to Portable Ontology Specifications. Knowledge Acquisition Vol. 6, No. 2, 1993

[HoHu02] Hofreiter, B., Huemer, C.: B2B Integration - Aligning ebXML and Ontology Approaches. Proc of the 1st EurAsian Conf. on Advances in Information and Communication Technology (EURASIA-ICT 2002). Springer LNCS, Vol. 2510, pp 339-349

[HoHu04] Hofreiter, B., Huemer, C.: Transforming UMM Business Collaboration Models to BPEL. Proc. of OTM Workshops 2004. Springer LNCS, Vol. 3292, (2004) 507-519

[HoHK06] Hofreiter, B., Huemer, C., Kim, J.-H.: Choreography of ebXML business collaborations. Journal of Information Systems and E-Business Management, Vol. 4, No. 3, Springer

[Li00] *Li, H.:* XML and Industrial Standards for Electronic Commerce, Knowledge and Information Systems Vol. 2, No. 4, 2000, pp. 487-497

[MaMi04] *Manola, F., Miller, E.:* RDF Primer, W3C Recommendation, February 2004, http://www.w3.org/TR/rdf-primer/

[McGH04] *McGuinness, D.L., van Harmelen, F.:* OWL Web Ontology Language Overview, W3C Recommendation, February 2004, http://www.w3.org/TR/owl-features/

[OmFe00] *Omelayenko, B., Fensel, D.:* Scalable Document Integration for B2B Electronic Commerce, http://www.cs.vu.nl/~borys/papers/OF_SII4.pdf

[Onto00] *Ontoprise,* B³ Semantic B2B Broker, http://www.ontoprise.de, 2000

[UN03] *UN/CEFACT TMG:* Core Components Technical Specification – Part 8 of the ebXML Framework, Version 2.01, 2003, http://www.untmg.org/dmdocuments/CCTS_v201_2003_11_15.pdf

[UN06] UN/CEFACT TMG: UMM Meta Model – Foundation Module, Candidate for 1.0, Final Working Draft, 2006-06-22, http://www.unece.org/cefact/umm/UMM_Foundation_Module.pdf

[W3C01] *W3C:* DAML+OIL (March 2001) Reference Description, W3C Note, December 2001, http://www.w3.org/TR/daml+oil-reference