Schema- and Ontology-Based XML Data Exchange in Semantic E-Business Applications

Jolanta Cybulka[1], Adam Meissner[1], Tadeusz Pankowski[1,2]
[1)] Institute of Control and Information Engineering,
Poznań University of Technology, Poznań, Poland
[2)] Faculty of Mathematics and Computer Science,
Adam Mickiewicz University, Poznań, Poland
{Jolanta.Cybulka | Adam.Meissner |
Tadeusz.Pankowski}@put.poznan.pl

Abstract

The problem of the semantics-preserving data transformations during their exchange by e-business applications is addressed. A method for automatic generation of schema mappings is proposed, which is based on the XML schema definition and the specification of the schema semantics via the relevant ontology.

1. Introduction

The semantic e-business is considered as an approach to managing knowledge for the coordination of e-business processes through the application of Semantic Web technologies including ontologies, knowledge representation methods, and intelligent software agents [Singh 2005]. These technologies provide means to design collaborative and integrative, inter- and intra-organizational business processes founded on efficient exchange of data among trusted business partners. Such an approach opens up new possibilities of cooperation among business partners, where their offers can be compared, even complex negotiations can be conducted electronically, and a contract can be drown up and fulfilled via an electronic marketplace [Pankowski 2005, Quix 2002].

Such a scenario requires the integration and the interoperation of applications (e.g. product databases, order processing systems) across multiple organizations. This integration is achieved through semantics-preserving data exchange, it means that there is a common understanding of exchanged data between their sender and receiver even though the data are structured using different terminologies and different formats. Such semantic integration of e-business processes must be carried out in a highly dynamic environment where companies enter the marketplace while some others drop out. Moreover, the system must be

adaptable for different environments, as the electronic marketplace covers many countries with different languages and different legal regulations.

Recently, much effort is being done in the area of semantics-based exchange of data [Arenas 2005, Doan 2004]. The proposed methods use the knowledge concerning the intended meaning of the whole schema (or its elements), which is obtained via some "revealing semantics" analysis of the schema, in most cases assisted by a human. The obtained semantics is usually formally specified in the form of ontology and helps to generate the data matches (matching discovery). There exist also the methods in which the matches are discovered by means of some heuristics and machine learning techniques [Rahm 2001].

The presented approach relays on the fact that many contemporary e-business applications use XML as a standard format for data exchange [XML 2004]. The data are often accompanied by their schemas written in XSD (XML Schema Definition) [XSD 2004]. Moreover, there exist some recommended XSDs for domain-oriented e-business applications, notably BMECat (www.bmecat.org), OpenTrans (www.opentrans.org) or XCBL (www.xcbl.org). For commonly accepted schemas of this kind, ontologies defining their semantics can be defined. It is convenient to embed such schema ontologies in a common foundational ontology, for it enables to obtain a set of correspondences between schema elements via some ontology-based reasoning process. The considered correspondences can be further used to generate semantics-preserving transformations of data. We propose a method to perform this task automatically using information provided by the considered schemas and the discovered set of correspondences.

In Section 2 a general structure of the process of schema- and ontology-based XML data exchange in the e-business scenario is discussed. Section 3 describes a method to generate executable schema mappings performing the mentioned transformations. In Section 4 the issues concerning ontology-based support for the schema matching are presented. Section 5 contains some final remarks.

2. Scenario of XML data exchange in e-business applications

The real-life e-business applications often process the XML data that are structured according to some standardized e-business schemas of catalogs and messages, such as BMECat, OpenTrans or XCBL [Rahm 2004]. Such catalogs of XSD are developed by various individual, national and public organizations, with the use of various languages, currencies, customs and national legal regulations. Thus the correspondences and mappings between them should be discovered and formally defined in the way that the (possibly automatic) semantics-preserving transformations between them would be possible.

The semantic integration of the considered e-business applications needs some "mutual understanding" of the data exchanged between them. It is defined

as follows. Assume that an application A_1 processes data structured under a schema S_1, which is semantically specified by means of the ontology O_1. Similarly, for an application A_2, a schema S_2 and an ontology O_2 are given. A data D_1 (the source instance) structured by A_1 under S_1 and O_1 is understandable by A_2 if D_1 can be converted into D_2 (the target instance) structured under S_2 that is specified by O_2. It means that there exists the semantic equivalency between data, based on the O_1 and O_2 alignment, i.e. there is a set of well-defined semantic functions converting D_1 to D_2 and these functions can be used to explain all steps of the conversion. The architecture of a system for such semantic data exchange is sketched in Fig. 1.

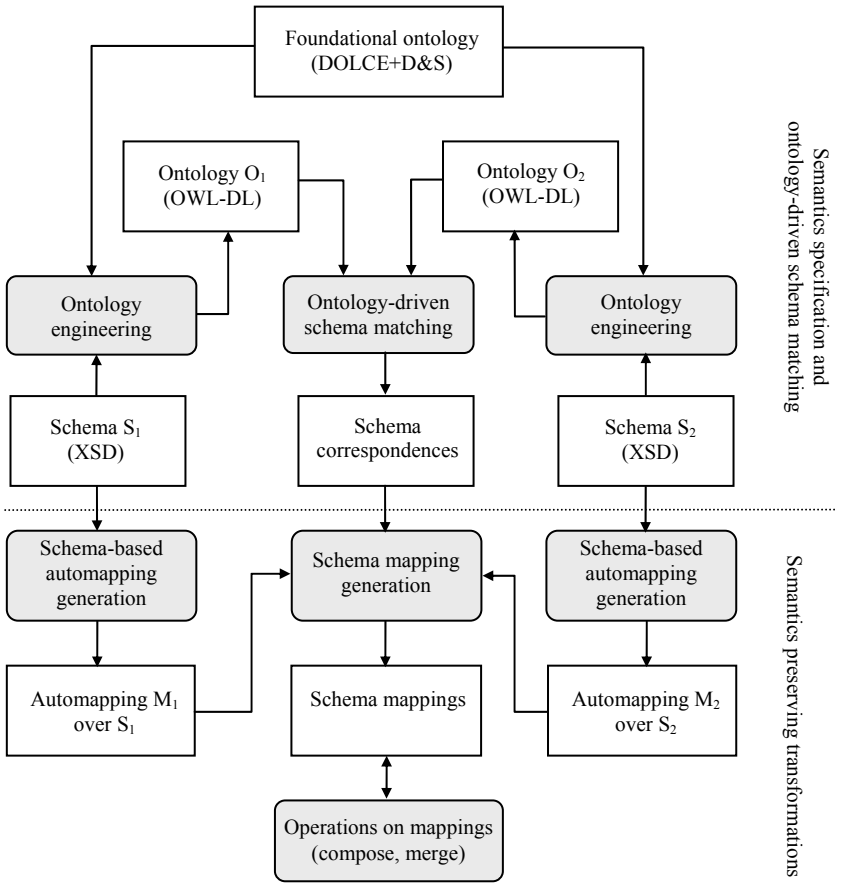

Figure 1. The structure of a semantic XML data exchange system.

The semantic data exchange process consists of the following two phases.

1. In the first phase, the ontologies O_1 and O_2 are specified by a knowledge engineer. The ontologies, which represent the semantics of schemas S_1 and S_2, are embedded in some shared foundational ontology by the use of its

ground ontological concepts and relations (see Section 4). Then the ontology-driven matching between the two schemas is done to obtain the correspondences between their elements (tags) or between sets of elements.

2. The obtained correspondences form the input to the schema mappings generation procedure in the second phase. This procedure uses also *automappings*, i.e. identity mappings over schemas. An automapping represents the schema and is generated based on constraints (keys and value dependencies) provided by the schema (see Section 3). The generated mappings between the schemas are "executable" in that they express not only relationships between schemas but also serve to transform the data instances. The considered mappings can be also composed and merged via formally defined operations.

3. Schema-based generation of mappings

In this section the problem of schema automappings and mappings generation is discussed, provided that the schemas are defined by means of XSD and a set of correspondences between schemas is given. Suppose that there are three applications A_1, A_2, and A_3 which exchange data structured according to the schemas S_1, S_2 and S_3, respectively (Fig. 2). If A_1 is to process data delivered by A_2 and/or A_3 then the data must be transformed appropriately so that it would be understandable by A_1, i.e. the data must be structured under S_1 and their semantics must be preserved. The transformation will be done by means of an executable mapping. Such a mapping can be generated automatically applying semantic correspondences between schemas, discovered using ontologies, and automappings over schemas (see Fig. 1). A schema automapping can be generated from constraints included in the schema. This process is illustrated by the example shown in Fig. 2.

The schema labels are: *supplier* (S) and *dealer* (D); *name* (N) and *address* (A) of the supplier; *name* (N) and *localization* (L) of the dealer; *part* (P); *identifier* (I), *price* (V) and *manufacturer* (M) of the part; address may be a text element (in S_1) or composed of *city* (C) and *street* (H) (in S_3). Elements labeled with R and K (in S_3) are used to join parts with their suppliers. I_2 and I_3 are the instances of S_2 and S_3, respectively. S_1 does not have any materialized instance.

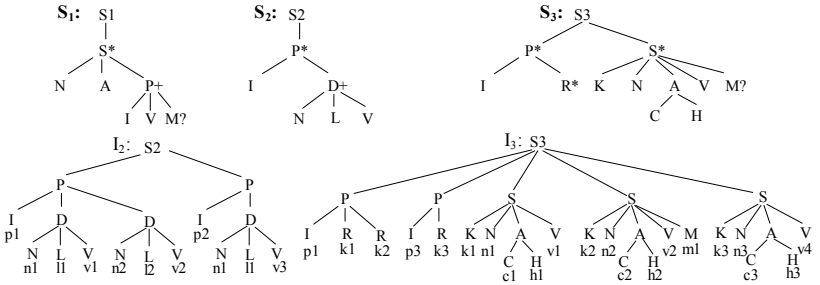

Figure 2. Tree representation of the XML schemas and their instances.

A schema, depicted in Fig. 2 in a form of the tree, can be defined by means of XSD. A fragment XSD for S_1 is (`<xs:valdep>` is a non-standard element):

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
 ...
 <xs:element name="S"><xs:complexType><xs:sequence>
    <xs:element name="N".../><xs:element name="A".../>
    <xs:element ref="P"/></xs:sequence></xs:complexType>
  <xs:key name="SKey">
    <xs:selector xpath="."/><xs:field xpath="N"/>
  </xs:key>
  <xs:valdep><xs:target name="A"/>
   <xs:function name="a"/><xs:source xpath="N"/>
  </xs:valdep>
 </xs:element>
 <xs:element name="P"><xs:complexType><xs:sequence>
    <xs:element name="I".../><xs:element name="V".../>
    <xs:element name="M".../></xs:sequence></xs:complexType>
  <xs:key name="PKey">
    <xs:selector xpath="."/><xs:field xpath="I"/>
  </xs:key>
  <xs:valdep><xs:target name="V"/>
   <xs:function name="v"/><xs:source xpath="I"/>
   <xs:source xpath="N" ref="SKey"/>
  </xs:valdep>
  <xs:valdep><xs:target name="M"/>
   <xs:function name="m"/><xs:source xpath="I"/>
  </xs:valdep>
 </xs:element>
</xs:schema>
```

From the above definition of the schema S_1, the automapping M_1 over S_1 can be generated, i.e. the identity mapping from the set of all instances of S_1 onto itself (the lines of the procedure are numbered for reference). Further on, a mapping M will be abbreviated as a quadruple $M = (G, \Phi, C, \Delta)$.

M_1 = **foreach** G_1 **where** Φ_1 **when** C_1 **exists** Δ_1=
(1) **foreach** $\$y_{S1}$ in /S1,$\$y_S$ in $\$y_{S1}$/S,$\$y_N$ in $\$y_S$/N, $\$y_A$ in $\$y_S$/A,
$\$y_P$ in $\$y_S$/P, $\$y_I$ in $\$y_P$/I, $\$y_V$ in $\$y_P$/V, $\$y_M$ in $\$y_P$/M,
(2) **where true**
(3) **when** $\$y_A$=a($\$y_N$), $\$y_V$=v($\$y_N$,$\$y_I$), $\$y_M$=m($\$y_I$)

exists
(4) $F_{/S1}()$ **in** $F_{()}()/S1$
(5) $F_{/S1/S}(\$y_N)$ **in** $F_{/S1}()/S$
(6) $F_{/S1/S/N}(\$y_N)$ **in** $F_{/S1/S}(\$y_N)/N$ **with** $\$y_N$
(7) $F_{/S1/S/A}(\$y_N,\$y_A)$ **in** $F_{/S1/S}(\$y_N)/A$ **with** $\$y_A$
(8) $F_{/S1/S/P}(\$y_N,\$y_I)$ **in** $F_{/S1/S}(\$y_N)/P$
(9) $F_{/S1/S/P/I}(\$y_N,\$y_I)$ **in** $F_{/S1/S/P}(\$y_N,\$y_I)/I$ **with** $\$y_I$
(10) $F_{/S1/S/P/V}(\$y_N,\$y_I,\$y_V)$ **in** $F_{/S1/S/P}(\$y_N,\$y_I)/V$ **with** $\$y_V$
(11) $F_{/S1/S/P/M}(\$y_N,\$y_I,\$y_M)$ **in** $F_{/S1/S/P}(\$y_N,\$y_I)/M$ **with** $\$y_M$

The line (1) defines variables. In (2) restrictions on variables are given (**true** indicates that there is not any restriction). The equalities in (3) reflect value dependencies specified in the `<xs:valdep>` of the XSD, where $\$y_A=a(\$y_N)$ says that the address of a supplier depends on its name, i.e. there exists a term of the form a(), which represents the mapping of suppliers' names on their addresses (the definition of the *a* function is unimportant). The same concerns the equations $\$y_V=v(\$y_N,\$y_I)$ and $\$y_M=m(\$y_I)$. The $F_{/S1/S}(\$y_N)$ expression is a Skolem term, where $F_{/S1/S}$ is the Skolem function name of type /S1/S. The value of the term is a node of the type /S1/S of the newly created XML data tree. The **in** operator indicates that its left-hand side belongs to the set of nodes denoted by the right-hand side expression. The text value of a leaf node is determined by the **with** operator. A Skolem function of the type P has as arguments the key paths of the element of type P. They are defined by means of the <xs:key> specification (e.g. I for /S1/S/P) and are inherited from their superelements (N for /S1/S/P). The key for a leaf element includes the leaf element itself, so its text value also belongs to the list of arguments (see e.g. $F_{/S1/S/P/M}(\$y_N,\$y_I,\$y_M)$).

Two new nodes are created (4), the root $r = F_{()}()$ and the node $n = F_{/S1}()$ of the outermost element of type /S1. The node *n* is a child of the type S1 of *r*. In (5) a new node *n′* is created for any distinct value of $\$y_N$, each such node has the type /S1/S and is a child of type S of the node created by $F_{/S1}()$ in (4). For any distinct value of $\$y_N$ (6), a new node *n″* of type /S1/S/N is created. Each such node is a child of type N of the node created by the invocation of $F_{/S1/S}(\$y_N)$ in (5) for the same value of $\$y_N$. Because *n″* is a leaf, it obtains the text value equal to the current value of $\$y_N$. Similarly for the rest of the specification. Analogously we can define automappings M_2 and M_3 for S_2 and S_3.

To generate mappings between schemas the correspondences between them are needed. Table 1 lists the correspondences denoted by σ, between maximal paths (i.e. paths leading from the root to a leaf) for S_1 and S_2, and for S_1 and S_3.

Table 1. The correspondences between schema paths.

$P \in S_1$	$\sigma(P) \in S_2$
/S1/S/N	/S2/P/D/N
/S1/S/A	/S2/P/D/L
/S1/S/P/I	/S2/P/I
/S1/S/P/V	/S2/P/D/V

$P \in S_1$	$\sigma(P) \in S_3$
/S1/S/N	/S3/S/N
/S1/S/A	f_A(/S3/S/A/C, /S3/S/A/H)
f_C(/S1/S/A)	/S3/S/A/C
f_H(/S1/S/A)	/S3/S/A/H
/S1/S/P/V	/S3/S/V

For example, σ(/S1/S/A) = f_A(/S3/S/A/C, /S3/S/A/H) denotes a 1:*n* correspondence (one path from S_1 corresponds to a pair of paths from S_3). The function f_A establishes a 1:1 correspondence between text values assigned to /S1/S/A, and a pair of values corresponding to /S3/S/A/C and /S3/S/A/H. If, for example, /S1/S/A:*"c1,h1"*, /S3/S/A/C:*"c1"*, and /S3/S/A/H:*"h1"*, then we have *"c1,h1"* = f_A(*"c1", "h1"*), where P:*v* indicates that a path P has a text value *v*.

The correspondences (Table 1) and automappings are input to generate mappings between schemas. Let $M_1=(G_1,\Phi_1,C_1,\Delta_1)$ and $M_2=(G_2,\Phi_2,C_2,\Delta_2)$ be automappings over S_1 and S_2, respectively. The mapping Map(S_2,S_1) from S_2 to S_1 is defined as follows:

$$M_{21} = \text{Map}(S_2,S_1) = (G_2,\Phi_2,C_1,\Delta_1)[\$y{:}P \rightarrow \$x{:}\sigma(P)],$$

where [\$y:P $\rightarrow$ \$x:σ(P)] denotes the replacement of any variable of type P (occurring in C_1,Δ_1) by the variable of type σ(P) defined in G_2. So, we have:

M_{21}= **foreach** $\$x_{S2}$ in /S2,$\$x_P$ in $\$x_{S2}$/P, $\$x_I$ in $\$x_P$/I, $\$x_D$ in $\$x_P$/D,
$\$x_N$ in $\$x_D$/N, $\$x_L$ in $\$x_D$/L, $\$x_V$ in $\$x_D$/V
where true
when $\$x_L$=a($\$x_N$), $\$x_V$=v($\$x_N$,$\$x_I$), $\$y_M$=m($\$x_I$)
exists
$F_{/S1}$() **in** F_0()/S1
$F_{/S1/S}$($\$x_N$) **in** $F_{/S1}$()/S
$F_{/S1/S/N}$($\$x_N$) **in** $F_{/S1/S}$($\$x_N$)/N **with** $\$x_N$
$F_{/S1/S/A}$($\$x_N$,$\$x_A$) **in** $F_{/S1/S}$($\$x_N$)/A **with** $\$x_A$
$F_{/S1/S/P}$($\$x_N$,$\$x_I$) **in** $F_{/S1/S}$($\$x_N$)/P
$F_{/S1/S/P/I}$($\$x_N$,$\$x_I$) **in** $F_{/S1/S/P}$($\$x_N$,$\$x_I$)/I **with** $\$x_I$
$F_{/S1/S/P/V}$($\$x_N$,$\$x_I$,$\$x_V$) **in** $F_{/S1/S/P}$($\$x_N$,$\$x_I$)/V **with** $\$x_V$
$F_{/S1/S/P/M}$($\$x_N$,$\$x_I$,$\$y_M$) **in** $F_{/S1/S/P}$($\$x_N$,$\$x_I$)/M **with** $\$y_M$

M_{21} converts any instance of S_2 into an instance under S_1. Moreover, all target constraints (keys and value dependencies) as well as all semantic correspondences between S_2 and S_1 are preserved.

The mapping M_{21} is an example of a *mapping composition*, $M_{12} = M_1 \bullet M_2$, where σ establishes a correspondence between *source*(M_2) and *source*(M_1).

Note, that while converting I_2 into I_{21}, the variable $\$y_M$ cannot be replaced. Then values of $\$y_M$ will be terms *"m(p1)"* and *"m(p2)"*, since $\$x_I$ will be bound to *"p1"* and *"p2"* within I_2 (Fig. 3(a)). These terms can be resolved while merging I_{21} with I_3 (for example *"m(p1)"* will be replaced by *"m1"* since I_3 provides information that the part *"p1"* is manufactured by *"m1"*).

To merge S_2 and S_3 under S_1, we define merging of mappings. Let $M_{21} = (G_1,\Phi_1,C_1,\Delta_1)$, $M_{31} = (G_2,\Phi_2,C_2,\Delta_2)$, M_{21} and M_{31} are defined using disjoint sets of variables, and *target*(M_{21}) = *target*(M_{31}). Then the merge of M_{21} and M_{31} is defined as the mapping:

$$M_{21} \cup M_{31} = (G_1 \cup G_2,\ \Phi_1 \wedge \Phi_2,\ C_1 \cup C_2,\ \Delta_1 \cup \Delta_2).$$

Intuitively, the result mapping consists of the union $G_1 \cup G_2$ of variable definitions over *source*(M_{21}) and *source*(M_{31}), and the conjunction $\Phi_1 \wedge \Phi_2$ of restrictions over these variables. C_1 and C_2 are defined over the same target schema and differ only in variable names. The same holds for Δ_1 and Δ_2. The result of merging I_2 and I_3 under S_1 is given in Fig. 3(b).

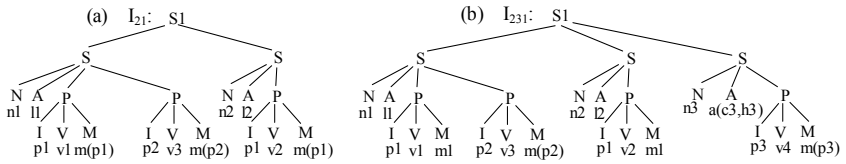

Figure 3. Results of $I_{21} = M_{21}(I_2)$ and $I_{231} = (M_{21} \cup M_{31})(I_2,I_3)$ transformations.

Note that values of some leaf elements (M) are missed and are replaced by terms following value dependencies defined in the schema. This convention forces some elements to have the same values. A term, like *"m(p1)"* can be resolved using other mappings (see Fig. 3(b)).

4. Ontology-Based Support for the Schema Matching

The possible correspondences between the schema elements (Fig. 1) relay on the *equivalence* and the *subsumption* relations between the ontological concepts representing schema elements. Roughly speaking, the equivalence holds between two identical concepts, while the subsumption between two concepts means that the first of them is more general then the latter (e.g. *Type* subsumes *GDT* and *HDT*, where GDT and HDT stand for gas device type and hydraulic device type, respectively). Generally, we consider the correspondence between a concept and a set of concepts, which has been previously denoted as 1:*n* (or 1:1 when the set consists of one element). Our approach does not comprise correspondences between sets of concepts since costs may exceed practical profits. The problem is computationally hard and many of its cases do not conform to the intended, intuitive meaning of the correspondence term.

The ontology is an engineering artifact which formally specifies a conceptualization, in the form of a structure consisting of the *concepts* and the *relations* which hold between them. The relations enable to specify the intended meaning of the concepts.

We consider the common shared (a reference) ontology, which is a structure representing the most general (domain independent) concepts, interrelated by the ground relations. It is referred to as the *foundational ontology* and presents the general view on reality, such as DOLCE (*Descriptive Ontology for Linguistic and Cognitive Engineering*) with the Descriptions&Situations (D&S) extension ([Masolo 2003], [Gangemi 2003]). For the sake of decidability and efficiency of inference methods we use only the parts, which can be expressed in Description Logics (DLs) [Baader 2003].

Recall the XML schemas from Fig. 2. The first step is to have some semantics attached to the schemas. The knowledge engineer tries to recall the semantics lying behind the tags names and the schema composition rules in the context of the DOLCE+D&S, and decides on the choice and the definition of the

ontological concepts and their embodiment in the reference ontology. The concepts will be defined in OWL DL [McGuinness 2004] and attached to the XML schema.

In DOLCE+D&S the conceptual specification of the reality may consist of the different (epistemic) layers, namely the "descriptions" (theories) and their models. The situation description should "satisfy" some real "situation". The relation of satisfaction is one of the ground relations.

The whole schema concerns a real business "situation", so the specialization of the *S-Description* concept is attached to it. Such a description is composed of: the action description (*CourseOfEvents*), the functional roles involved in the action (*AgentiveFunctionalRole* and *Non-agentiveFunctionalRole*) and the parameters of the situation (*Parameter*). The standard ground relations connect the components of the situation description (Fig. 4). In particular, we assume the existence of the *Selling* concept which specializes the *BusinessActivity* concept, which in turn is a specialization of the *CourseOfEvents*.

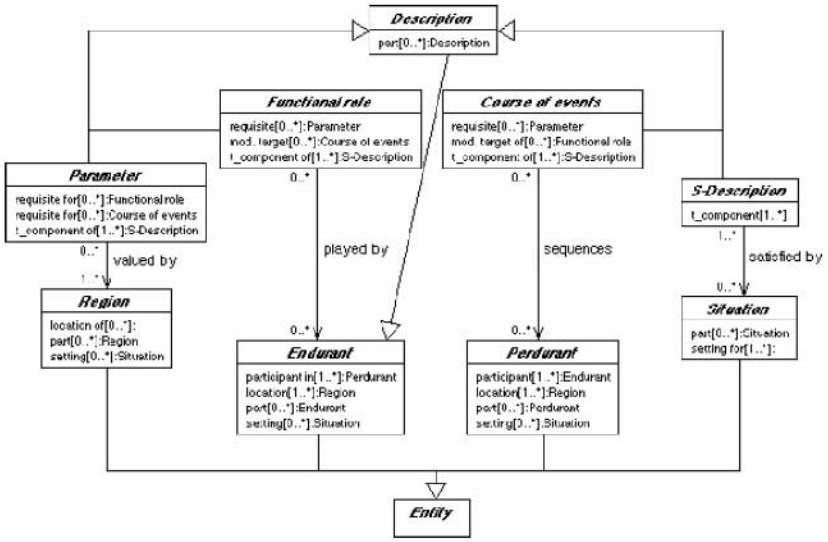

Figure 4. The top-level of DOLCE+D&S in UML [Masolo 2003, p.98].

The *Dealer* and the *Supplier* occurring in the schema are modeled as the (subject of the activity) *AgentiveFunctionalRoles* which are linked to the *Selling* by the ground *modality_target* relation. The selling activity has also the object of the activity, the *Part* element, which is the specialization of the *Non-agentiveFunctionalRole* (also linked to itthe *Selling* by the *modality_target*). The *Selling* situations have *Parameters*, i.e. some XML elements connected to the *Supplier*, the *Dealer* and the *Part*. All the *Parameters* can be assigned values coming from some *Regions* by means of the *valued_by* ground relation. The *requisite* relation binds the *Parameters* to the functional roles and the courses of events. The situation description components can be also linked to the

ontological entities that constitute the real situations, not mentioned here (see Fig. 4).

The semantics of various schemas may be defined by different knowledge engineers. Hence, the same concepts may be defined under different names and with slightly different descriptions. Therefore, the ontology O_1 contains the following definition of the concept (class) *Supplier*:

```
<owl:Class rdf:ID="Supplier">
       <owl:unionOf rdf:parseType="Collection">
           <owl:Restriction>
             <owl:onProperty rdf:resource="&ONT;supplies" />
               <owl:allValuesFrom rdf:resource="#GDT" />
           </owl:Restriction>
           <owl:Restriction>
             <owl:onProperty rdf:resource="&ONT;supplies" />
               <owl:allValuesFrom rdf:resource="#HDT" />
           </owl:Restriction>
        </owl:unionOf>
</owl:Class>
```

The ontology O_2 might define a concept *Dealer* as follows:

```
<owl:Class rdf:ID="Dealer">
   <owl:intersectionOf rdf:parseType="Collection">
      <owl:Restriction>
            <owl:onProperty rdf:resource="&ONT;supplies" />
               <owl:allValuesFrom>
                 <owl:unionOf rdf:parseType="Collection">
                   <owl:Class rdf:about="#GDT />
                    <owl:Class rdf:about="#HDT />
                  </unionOf>
               </allValuesFrom>
       </owl:Restriction>
       <owl:Restriction>
           <owl:onProperty rdf:resource="&ONT;supplies" />
            <owl:maxCardinality
                  rdf:datatype="&xsd;nonNegativeInteger"> 1
            </owl:maxCardinality>
       </owl:Restriction>
   </owl: intersectionOf >
</owl:Class>
```

The description *Supplier* symbolizes a set of individuals, who supply gas devices (GDT) only and also those who supply only hydraulic devices (HDT), while the concept *Dealer* contains individuals, who supply only one type of devices, which are either gas devices or hydraulic devices. The role *supplies* specializes the relation *requisite*. The descriptions of concepts GDT and HDT are identical.

In order to formally specify the correspondence relationships between XML schemas we define the subsumption and the equivalence in terms of DLs. It should be remarked, that a concept in DLs is interpreted as a set of individuals and the expression $C \sqcup D$ denotes a union (in the sense of the set theory) of concepts C and D. Let $CS = \{C_1, \ldots, C_n\}$ be a set of concepts C_i for $i = 1, \ldots, n$ and let $\sqcup CS = C_1 \sqcup \ldots \sqcup C_n$.

1. A concept *D* (*subsumer*) *subsumes* a set of concepts *CS* (*subsumee*) iff $\sqcup CS$ is a subset of *D*.
2. A set of concepts *CS* and a concept *D* are *equivalent* iff $\sqcup CS$ subsumes *D* and *D* subsumes $\sqcup CS$.

The subsumption between concepts can be established by inference methods described in ([Baader 2003]). For example, it can be concluded that the concept *Supplier* defined in the ontology O_1 subsumes the concept *Dealer* from the ontology O_2.

A subsumption not always conforms to the intuitive meaning of the correspondence intended by the user. Thus, the decision about including the particular case of subsumption to the correspondence relation in general cannot be made automatically. Moreover, when looking for the schema element, which corresponds "by subsumption" to the given element, one expects that the subsumer is possibly the most specific (i.e. the least) concept. In other words, the subsumer should directly succeed the concept modeling the considered element in the order defined by the subsumption relation. In case of the set of concepts, the subsumer should be the least common subsumer [Baader 2003] of all elements included in the set. This can be retrieved from the *subsumption hierarchy*, also called the *taxonomy*. The taxonomy is a graph, which represents the partial ordering (by subsumption) of the given set of concepts. It may be constructed in many ways, assuming that a procedure of testing for the subsumption between two concepts is given. A good introduction to algorithms of computing the subsumption hierarchy contains [Baader 1992]. Many of them introduce (for technical reasons) the element $\top$ (*top*) to the taxonomy, which may be absent in the input set of concepts. This element represents the most general concept, that is the subsumer of all considered concepts.

Summing up, we propose the simple method of establishing correspondence relationships between elements of two XML schemas, say S1 and S2. In the first step the taxonomy *T* is created for the set of concepts representing all elements of the schema S1 and S2. Then, the taxonomy *T* is taken as an input to the algorithm *FindCorresp*, which constructs the correspondence relation *Cor*. We assume that the symbol $F(C)$ for every concept *C* from the taxonomy *T* denotes a set of concepts directly subsumed by *C* and belonging to the other schema than *C* belongs to. It should be observed that *C* is in general the least common subsumer of $I(C)$.

Algorithm *FindCorresp*.
Input: *T* – the taxonomy of elements of the schema S1 and S2.
Output: *Cor* – the correspondence relation between elements of S1and S2.

begin

1. Let $Cor := \emptyset$.
2. For every concept *C* from *T*, such that $C \neq \top$ and $F(C) \neq \emptyset$ perform 3 then end.

3. If C is subsumed by $\sqcup F(C)$ then let $Cor := Cor \cup (C, F(C))$ else let $Cor := Cor \cup Ask(C, F(C))$.

end.

The condition from the step 3 expressing that C is subsumed by $\sqcup F(C)$ implies the equivalence between C and $\sqcup F(C)$ since the subsumption in the opposite direction follows from the definition of the set $F(C)$. Therefore, if C and $F(C)$ are equivalent, then the pair $(C, F(C))$ may be added to the relation Cor. In the other case C and $F(C)$ are related only by subsumption, so their correspondence has to be verified by a human. Hence, the function *Ask* is called, which asks the user whether the first argument (C) corresponds to the second one, that is $F(C)$. If the answer is positive then the function returns a pair $(C, F(C))$, otherwise the value of the function *Ask* is $\varnothing$.

In particular case one may be interested in finding correspondences between leaves of XML schemas only. In this situation, the set of concepts for which the taxonomy is constructed (phase 1) should be restricted to leaves of the regarded schemas.

5. Final remarks

The problem of schema mapping is considered, which occurs in XML data exchange among e-business applications. Our solution to this problem relies on the automatic generation of semantics-preserving schema mappings. We discussed how automappings could be generated using keys and value dependencies defined in XML Schema. Constraints on values can be used to infer some missing data. Mappings between two schemas can be generated automatically from their automappings and correspondences between schemas. Automappings represent schemas, so operations over schemas and mappings can be defined and performed in a uniform way. We also outline the idea, how the ontological knowledge describing the semantics of XML schemas may be applied for finding correspondences between elements of different schemas.

6. References

[Arenas 2005] Arenas, M., Libkin, L., XML Data Exchange: Consistency and Query Answering, PODS Conference, 2005, pp. 13-24.

[Baader 1992] F. Baader, E. Franconi, B. Hollunder, B. Nebel, H.J. Profitlich, An empirical analysis of optimization techniques for terminological representation systems, or: Making KRIS get a move on, Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR'92), 1992, pp. 270–281.

[Baader 2003] F. Baader, D.L. McGuinness, D. Nardi, P.F. Patel-Schneider (eds), The Description Logic Handbook: Theory, implementation, and applications, Cambridge University Press, 2003.

[McGuiness 2004] D.L. McGuinness, M.K. Smith, C. Welty, OWL Web Ontology Language Guide, http://www.w3.org/TR/owl-guide/, 2004.

[Doan 2004] A. Doan, N.F. Noy, A. Halevy, Introduction to the Special Issue on Semantic Integration, SIGMOD Record, 33(4), 2004, pp. 11-13.

[Gangemi 2003] A. Gangemi, P. Mika, Understanding the Semantic Web through Descriptions and Situations, ODBASE03, Berlin, 2003.

[Masolo 2003] C. Masolo, S. Borgo, A. Gangemi, N. Guarino, A. Oltramari, WonderWeb Deliverable D18; Ontology Library (final), http://wonderweb.semanticweb.org/deliverables/D18.shtml, 2003.

[Noy 2004] N.F. Noy, Semantic Integration: A Survey of Ontology-Based Approaches, SIGMOD Record, 33(4), 2004, pp. 65-70.

[Pankowski 2005] Pankowski T., Integration of XML Data in Peer-To-Peer E-commerce Applications, IFIP I3E, Springer, New York, 2005, pp. 481-496.

[Quix 2002] Quix, C., Schoop, M., Jeusfeld, M. A., Business Data Management for B2B Electronic Commerce. SIGMOD Record, 31(1), 2002, pp. 49–54.

[Rahm 2001] Rahm, E., Bernstein, P. A., A survey of approaches to automatic schema matching, The VLDB Journal, 10(4), 2001, pp. 334-350.

[Rahm 2004] Rahm E., Do H.-H., Massmann S., Matching Large XML Schemas, SIGMOD Record, 33(4), 2004, pp. 26-31.

[Singh 2005] R. Singh, L.S. Iyer, A.F. Salan, The Semantic E-Business Vision, Communications of the ACM, Vol. 48, No. 12, 2005, pp. 38-41.

[XML 2004] XML 1.0, W3C, http://www.w3.org/TR/REC-xml.

[XSD 2004] XML Schema, W3C, http://www.w3.org/TR/xmlschema-0