OntoLiFT Prototype

WonderWeb: Ontology Infrastructure for the Semantic Web

Raphael Volz[1,2], Daniel Oberle[1], Steffen Staab[1], Rudi Studer[1,2]

[1]University of Karlsruhe
Institute AIFB
D-76128 Karlsruhe
email: {lastname}@aifb.uni-karlsruhe.de

[2]FZI - Research Center for Information Technologies
Haid-und-Neu-Strasse 10-14
D-76131 Karlsruhe
email: {lastname}@fzi.de

Identifier	Del 11
Class	Deliverable
Version	1.2
Date	13-01-2003
Status	Final
Distribution	Internal
Lead Partner	AIFB

WonderWeb Project

This document forms part of a research project funded by the IST Programme of the Commission of the European Communities as project number IST-2001-33052.

For further information about WonderWeb, please contact the project co-ordinator:

Ian Horrocks
The Victoria University of Manchester
Department of Computer Science
Kilburn Building
Oxford Road
Manchester M13 9PL
Tel: +44 161 275 6154
Fax: +44 161 275 6236
Email: wonderweb-info@lists.man.ac.uk

Contents

Executive Summary

This deliverable describes OntoLift. It aims at leveraging existing schema structures as a starting point for developing ontologies for the Semantic Web.

A huge effort has been invested in the development of schema structures for existing information systems, such as XML-DTD, XML-Schema, relational database schemata or UML specifications of object-oriented software systems. The LiFT tool semi-automatically extracts light ontologies from such legacy resources. We restrict our attention to the most important ones, namely the W3C schema languages for XML: Document Type Definitions (DTDs), XML Schema and relational database schemata. We also provide a preliminary translation from UML-based software specifications to ontologies.

1 Introduction

This deliverable describes the methodology, architecture and design of OntoLift, which takes a dual role in WonderWeb: First, it provides means for bootstrapping ontologies by extracting the conceptualizations inherent in schemas which underly existing data and information systems. Second, it will allow to access data stored according to those schemas in existing information systems through ontologies.

This document focuses on the first role. Since the diversity and variety of possible data sources is unaccountable, we focus on three prominent data formats: XML documents, object-oriented specifications and relational database schemas.

The latest versions of the prototype can be accessed via the Karlsruhe Semantic Web and Ontology Tool Suite (KAON) website at http://kaon.semanticweb.org.

The document is structured as follows. Section 2 describes the extraction of ontologies from relational database schemas. Section 3 describes the extraction of ontologies from XML schema languages and section 4 details the extraction of ontologies from a representative object specification language: the OMG UML standard.

Each section is structured in a similar fashion. First, the central features of the data model are described and aligned with the entities provided by Web ontology languages. Second, the transformation heuristics are presented. Third, central aspects of the implementation are presented together with an outlook for further applications of the results and related work.

2 Relational Databases

This section describes the extraction of ontologies from existing logical database schemas. The approach presented here tries to reverse the well-known mapping process from conceptual database schemes (such as the Entity Relationship Model) to the logical schema found in operational relational databases. The results were presented in [28].

2.1 Prerequisites

Relational databases are a predominant component in the modern information system landscape. Almost all modern information systems which need to deal with non-minuscule amounts of data build upon such databases as a storage and data processing component.

Data is stored based on logical schemas. Those schemas already provide some description about the universe of discourse (UoD) in which the given information system operates. Hence, it is promising to bootstrap the ontology development process with relational schemas.

However, one may not expect to extract very complex ontologies that meet high quality demands. This is due to the fact that the description provided by a relational schema usually incorporates many choices that are conceptually irrelevant for the description of

the UoD, e.g. choices made for performance reasons. Hence the conceptualization principle [31] is typically violated. Second, the relational data model does not suffice for complex descriptions. This lack of sufficiently powerful construction mechanisms leads to so-called overspecification, i.e. the introduction of extra entities to capture certain facts.

2.2 The relational model

The underlying model of relational databases is the relational model. We extend the usual formal definition of the relational model (e.g. in [26]) with additional constructs typically found in SQL-DDLs, i.e. constructs which allow to state inclusion dependencies [14].

Definition 2.2.1 *A relational Model M is a 8-tuple* $(R, A, T, I, att, key, type, notnull)$ *with:*

1. *A finite set R called Relations*
2. *A finite set A called Attributes*
3. *A function* $att : R \rightarrow 2^A$ *which defines the attributes contained in a specific relation* $r_i \in R$
4. *function* $key : R \rightarrow 2^A$ *that defines which attributes are primary keys in a given relation used to identify a certain tuple in the relation, thus* $key(r_i) \subseteq att(r_i)$ *must hold*
5. *A set* T *of atomic data types*
6. *A function* $type : R \rightarrow T$ *that states the type of a given attribute.*
7. *A function* $notnull : R \rightarrow 2^A$ *which states those attributes of a relation which have to have a value.*
8. *A set of inclusion dependencies I where each element has the form* $((r_1, A_1), (r_2, A_2))$ *with* $r_1, r_2 \in R$, $A_1 = \{a_{11}, a_{12}, ..., a_{1n}\}$, $A_2 = \{a_{21}, a_{22}, ..., a_{2n}\}$, $A_1 \subseteq att(r_1)$ *and* $A_2 \subseteq att(r_2)$, $\mid A_1 \mid = \mid A_2 \mid$ *and* $type(a_{1i}) = type(a_{2i})$

Remark 2.2.2

1. *We will refer to* r_1 *as domain relation and* r_2 *as range relation.*
2. I_c *denotes the transitive closure of I*

The reader may note that SQL-DDLs are typically more expressive than relational algebra. For instance, it is usually possible to specify further constraints (such as DEFAULT and NOT NULL). Default values for datatypes are not supported in Web ontology languages and therefore ignored. The NOT NULL constraint is translated to the function notnull. Please note that SQL enforces automatically that $\forall r_i \in R : key(r_i) \subseteq notnull(r_i) \subseteq att(r_i)$.

Due to the static nature of ontologies the dynamic aspects found in SQL-DDLs cannot be converted, thus triggers, referential actions (like ON UPDATE etc.) and assertions cannot be mapped.

In SQL-DDLs it is also possible to specify referential integrity constraints, by means of so-called foreign keys. This information is especially useful for the mapping process as it indicates associations between database relations. SQL referential integrity constraints reinforce the view that inclusion dependencies [14] are valid at all times.

With respect to inclusion dependencies, a pitfall in the translation process becomes apparent: Usually the database designer should express associations between database relations by means of foreign keys to make these semantics explicit. However, the processing of this semantics is not supported by its definition. The combination of information supported via such associations has to be created manually by stating appropriate joins of tables in the queries to the database. Hence, those associations remain unspecified in many cases. The appropriate semantics could only be extracted by analyzing the queries sent to the database. However, the latter is not feasible since running information systems would have to be altered to track the queries issued by the system. We therefore allow users to additionally specify foreign keys a-posteriori.

2.3 Lifting Process

The lifting process tries to capture the semantics of the database by translating relations and attributes to concepts and roles in the ontology. This translation is performed via a set of translation rules specified in the following and inspired by [3].

The translation creates all possible ontological entities, detects overspecifications, i.e. auxiliary relations, in the original schema and eventually removes redundant information.

The translation is performed in two phases. In the first phase concepts are established. In the second phase roles of concepts are created from the database.

The translation rules are checked in sequential order on each relation. The first rule whose preconditions apply is then chosen for the respective relation. This is due to the fact that mapping rules might be ambiguous. Eventually, the lifting process should be influenced and directed by the user to choose the appropriate rule for individual relations. However, this is currently not implemented in the prototype.

2.4 Translation rules

All mapping rules are introduced with formal lists of preconditions and result in the appropriate ontological entities. In the following we use the auxiliary functions:

- $concept : R \rightarrow C$ to denote the relation with a concept is associated with and.
- $typetrans : T \rightarrow D$ that transforming relational data types into the appropriate XML schema datatypes[1].

To illustrate the mapping rules we refer to an example schema that stores data in the university teaching domain. This schema is depicted in Figure 1.

[1]This currently defaults into rdf:Literal since no Wonderweb system supports XML Schema datatypes

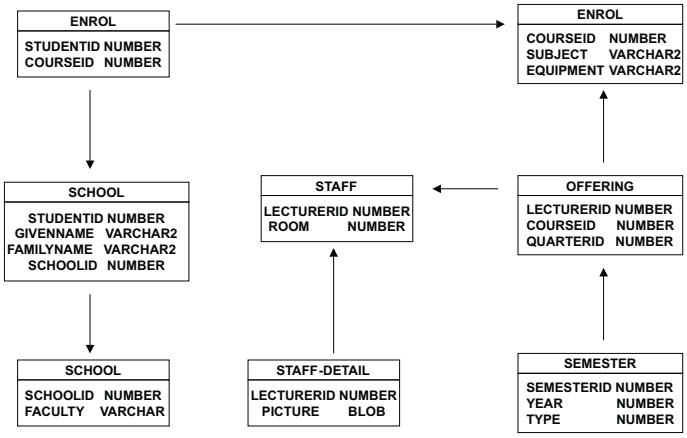

Figure 1: Example database schema

2.4.1 Concept creation

Certain database relations, e.g. ENROL, are only defined to express (n:m) associations between two other tables. Typically, this type of auxiliary relation is characterized by the fact that it contains only two attributes, which are both primary keys and foreign keys to two other relations.

Translation Rule 2.4.1 ((n:m) Association) *Auxiliary relations used to specify (n:m) associations may be identified via the following conditions:*

- $att(r_i) = key(r_i)$
- $A_1 \subset att(r_i), A_2 \subset att(r_i)$
- $A_1 \cup A_2 = att(r_i)$
- $A_1 \cap A_2 = \emptyset$
- $((r_i, A_1), (r_j, key(r_j))) \in I$
- $((r_i, A_2), (r_k, key(r_k))) \in I$

As a result no concept is created. Instead a similar rule triggers the creation of a role on both concepts c_j, c_k which are mutually inverse to each other[2].

In certain cases information that logically corresponds to one entity in the universe of discourse is distributed across several tables for performance reasons. This is for example the case if one of the attributes is storage-intensive and optional. To optimize the clustering behavior of the database such attributes are often stored in separate relations together with the primary key of the main relation, e.g. the STAFF-DETAIL relation.

Translation Rule 2.4.2 (Information distribution) *Information distribution may be detected with the following heuristic:*

[2] where $c_x = concept(r_x)$

- $((r_i, key(r_i)), (r_j, key(r_j)) \in I$

As a result no concepts are created. Instead a similar rule triggers the aggregation of attributes of both relations and their conversion to roles on only one concept.

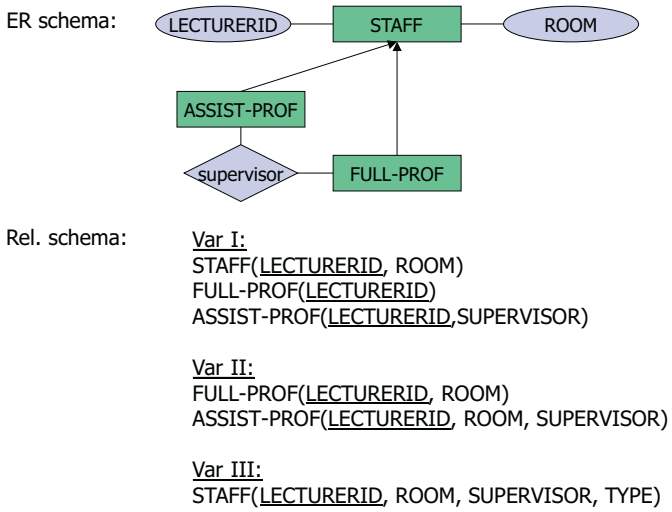

Figure 2: Specialization in Extended Entity Relationship Diagrams

This heuristic is slightly problematic since it also covers one of the well-known mapping procedures for translating entity hierarchies in EER models to relational schemas. Hence, a similar heuristic could alternatively lead to two concept definitions and a subsumption relation between those concepts. Figure 2 depicts such a situation. Here the entity STAFF has several specialized entities ASSIST-PROF and FULL-PROF. Usually this may be translated to relational schemas via the following principles [2]:

1. Create a relation for the super entity and relations for each sub entity, such that they contain all attributes of the sub entity and the primary key of the super entity.
2. If the specialization is total (there is no instance of the super entity, only sub entities are instantiated): Create a relation for all sub entities only and copy the attributes defined for the super entity
3. If the specialization is disjoint (an instance may only be instance of one entity): Create only one relation that contains the attributes defined for all entities and makes those attributes optional. Add an extra attribute to explicitly state the type

This clearly shows that the semantic intention behind a given relational structure cannot be captured fully by OntoLift. For example, heuristic 2.4.2 cannot decide which relation represents the sub entity and which one represents the super entity[3].

Translation Rule 2.4.3 *Our general heuristic is that each relation is converted into a concept.*

This heuristic is applied when no other rule could be applied.

[3]This could only be done by comparing the set of keys of the involved relations against each other and detecting which set is contained in the other. Hence this is extensionally defined and mutable.

2.4.2 Role creation

Attributes vs. associations The OntoLift tool distinguishes between attributes and associations[4]. The tool cannot decide whether a given attribute is only defined for organizational purposes, such as many auto-incremented primary keys, which do not carry any real meaning beyond tuple identification[5]. Hence, all attributes are converted to datatype properties[6]. The domain of attributes is naturally the concept, which has been created for the relation upon which the attribute is defined.

Associations between database relations, which are expressed via foreign keys (and constitute inclusion dependencies), are translated to object properties. Hence, inclusion dependencies are translated to object properties. The domain concept of an object property corresponds to the domain-relation of the inclusion dependency. The range concept corresponds to the range-relation of the inclusion dependency, respectively.

Cardinalities Each relational attribute has by default a maximum cardinality of one and a minimum cardinality of zero. If the attribute is not nullable ($a \in notnull(r_i)$) then the minimum cardinality is altered to 1. For foreign keys the maximum cardinality is unlimited, unless the foreign key is part of the primary key of the relation.

If a relation constitutes a (n:m)-association between to other database relations, no further action with respect to cardinalities has to be taken, since the relation is not translated to a concept.

Translation rules A default rule will be applied if no other rules are applicable. Hence all transaction rules are applied in the following order:

Translation Rule 2.4.4 ((n:m) association) *The first heuristic takes care of (n:m) associations between database relations. The preconditions for the application of this heuristic are:*

- $A_1 \subset att(r_i), A_2 \subset att(r_i)$
- $att(r_i) = key(r_i)$
- $A_1 \cup A_2 = att(r_i)$
- $A_1 \cap A_2 = \emptyset$
- $((r_i, A_1), (r_j, key(r_j)) \in I$
- $((r_i, A_2), (r_k, key(r_k)) \in I, r_j \neq r_k$
- $c_j = concept(r_j)$
- $c_k = concept(r_k)$

[4]In OWL terminology: datatype properties and object properties.

[5]The role of the latter is provided by URIs in the Semantic Web, hence this information would no longer be needed.

[6]Please note, that the current prototype is limited to rdf:Literal

As a result two new roles, $role_{i1}$ and $role_{i2}$, are created which are mutually inverse to each other , and have c_j and c_k as domain and range respectively. This leads to $inverse(role_{i1}, role_{i2}), domain(role_{i1}, c_j), domain(role_{i2}, c_k), range(role_{i1}, c_k)$ as well as $range(role_{i1}, c_j)$.

Translation Rule 2.4.5 ((1:1) association) *This heuristic captures (1:1) associations between database relations. The preconditions for the applicability of this heuristic are:*

- $c_i = concept(r_i)$
- $c_j = concept(r_j)$
- $((r_i, key(r_i)), (r_j, key(r_j)) \in I$
- $((r_j, key(r_j)), (r_i, key(r_i)) \in I$

As a result two new roles, $role_{i1}$ and $role_{i2}$, are created which are mutually inverse to each other , and have c_i and c_j as domain and range respectively. This leads to $inverse(role_{i1}, role_{i2})$, $domain(role_{i1}, c_i)$, $domain(role_{i2}, c_j)$, $range(role_{i1}, c_j)$ as well as $range(role_{i1}, c_i)$. With respect to cardinalities the cardinality of both roles is altered to one.

Translation Rule 2.4.6 ((1:m) association) *This heuristic treats one to many associations, which are constituted by foreign keys. The precondition for the application of this heuristic are:*

- $c_i = concept(r_i)$
- $c_j = concept(r_j)$
- $A_1 \subseteq att(r_i)$
- $((r_i, A_1), (r_j, key(r_j)) \in I$

As a result two new roles, $role_{i1}$ and $role_{i2}$, are created which are mutually inverse to each other, and have c_i and c_j as domain and range respectively. This leads to $inverse(role_{i1}, role_{i2}), domain(role_{i1}, c_i), domain(role_{i2}, c_j), range(role_{i1}, c_j)$, as well as $range(role_{i1}, c_i)$. With respect to cardinalities the minimum cardinality of $role_{i1}$ may be altered to one, if $A_1 \cup nullable(r_i) = \emptyset$.

Translation Rule 2.4.7 (Role Grouping) *This heuristic groups the attributes that are distributed in several relations and complements rule 2.4.2. The preconditions for the applicability of this heuristic are:*

- $\neg\exists c_i = concept(r_i)$
- $\exists c_j = concept(r_j)$
- $((r_i, key(r_i)), (r_j, key(r_j)) \in I$

For all attributes $A = att(r_i) \setminus key(r_i)$ new roles with domain concept c_j are created.

Translation Rule 2.4.8 (Default) *An attribute is translated to a role whose range has the corresponding XML Schema type*[7] *and whose domain is the concept created for the relation for which the attribute defined.*

2.5 Prototype Implementation

We avoid parsing DDL definitions due to the different SQL DDL dialects imposed by individual databases. Instead the implementation is based on the DatabaseMetaData interface offered by the Java JDBC standard. This interface offers a standardized connection to database metadata such as schemas. Unfortunately, it is not always fully implemented in the drivers offered by database vendors. However, we expect that the support for this interface will grow in the future.

When translating the schema into a RDF representation naming issues become apparent, since all identifiers used in a RDF file must be valid Uniform Resource Identifiers (URI). Users have to specify a URI prefix. This prefix is then used for all created entities. Concept identifiers are of the form `<prefix>/<schema>/<relation>`. Role identifiers are of the form `<prefix>/<schema>/<relation>/<attribute>`. Inverse roles are identified by `<prefix>/<schema>/<relation>/<attribute>/inverse`. In order to promote the readability in visual ontology editors the original relation / attribute names are stored as RDF labels.

Limitations Datatypes are not yet implemented by the prototype, the demonstrator will use a suitable translation matrix from SQL datatypes to the XML Schema datatypes supported by both RDF Schema and OWL. The prototype itself is currently limited to lightweight Web ontologies stated in RDF Schema. Hence, cardinalities and inverse relations are not implemented.

2.6 Further Applications

In today's Web, business applications have moved away from static, fixed web pages to those that are dynamically generated at the time of user requests. This kind of web sites are typically realized using relational databases. Such data-intensive web sites have numerous benefits, e.g. a simplified maintenance of the web design (due to complete separation between data and layout), the automated updating of web content etc. Nevertheless, several problems exist, the most prominent ones are probably [11]:

1. Data-intensive web sites form an invisible web. Search engine crawlers usually do not read dynamically generated URLs, thus pages are not included in search engine indexes. Consequently they are invisible.

2. The content of the database-driven web sites is not machine-understandable - information presented by using HTML is intended for user consumption only. Consequently they are not a part of the Semantic Web.

[7] Not in the prototype, here only rdf:Literal is created

Both problems may be overcome using OntoLift. The extracted ontologies can form the conceptual backbone for metadata annotations that are automatically created from the database contents. Hence, the tedious process of providing metadata, called semantic annotation [18], can be automated inexpensive and fast. This metadata may then be used by specialized search engines and becomes machine-understandable due to the explicit conceptualization provided by ontologies.

2.7 Related Work

There are very few approaches investigating the transformation of a relational model into an ontological model. The most similar approach to our approach is the project Infosleuth [19]. In this project an ontology is built based on the database schemas of the sources that should be accessed and is refined based on user queries. However, the presented techniques for creating ontologies are only suggestions to so-called domain experts, which have to create the ontology manually. Moreover, the semantic characteristics of the database schema are not fully analyzed.

More work has been addressed on the issue of explicitly defining semantics in database schemas [25, 4], extracting semantics out of database schema [7, 32, 17] and transforming a relational model into an object-oriented model [32, 3] which is close to an ontological theory. Rishe [25] introduces semantics of the database as a ”means” to closely capture the meaning of user information and to provide a concise, high-level description of that information. In [3] an interactive schema migration environment that provides a set of alternative schema mapping rules is proposed. In this approach, which is similar to our approach on the conceptual level, the re-engineer repeatedly chooses an adequate mapping rule for each schema artefact. However, this stepwise process creates an object-oriented schema, therefore cardinalities are not discussed.

3 XML

This section presents the translation of XML schema languages to ontologies. XML is a meta language for creating markup languages. Several XML schema languages have been devised. To generalize our results we present a translation based on so-called tree grammars. The proposed XML schema languages can easily be mapped to such grammars [22]. The prototype itself currently only supports DTDs and the W3C XML Schema standard

3.1 Prerequisites

Usually markup languages are intended to allow the exchange of structured documents. Nowadays it is increasingly being used as a means for exchanging data. This is a purpose for which XML has not been devised for and is not suited perfectly. Hence, XML schema languages contain many features that are irrelevant for data interchange and concentrate on document structural aspects instead of data structural aspects. Many aspects, e.g. the

order of markup tags, cannot be described in ontology languages and therefore remain untranslated.

Several languages for writing schemas have been proposed in the past, e.g. DTD [5], XML-Schema [30], DSD [20] and Relax NG [8]. To overcome the specifics of those individual languages we build our mappings on a formal framework, namely regular tree grammar theory (cf. [29] or [12]). This allows to generalize the mapping approach. An important benefit of this approach, beyond generalization, is the close correspondence to XML query languages, since XQuery is based on such tree grammars. Hence, our approach may easily be extended towards actively querying XML documents based on such schemas through the translated ontologies.

3.2 Tree Grammars

Definition 3.2.1 (Regular Tree Grammar) *A regular tree grammar (RTG) is a 4-tuple* $G = (N, T, S, P)$ *where:*

- N *is a finite set of non-terminals*
- T *is a finite set of terminals*
- S *is a set of start symbols, with* $S \subset N$
- P *is a finite set of production rules of the form* $X \rightarrow a\ r$*, where* $X \in N, a \in T$ *and* r *is a regular expression of* N*.*

We call X is the left-hand side of a production rule, whereas $a\ r$ is called the right-hand side, and r is the content model of a production rule.

Example 3.2.2 *The following grammar* $G_1 = (N, T, S, P)$ *is such a regular tree grammar:*

```
N = {Doc, Para1, Para2, PCDATA}
T = {doc, para, pcdata}
S = {doc}
P = {Doc -> doc (Para1, Para2*), Para1 -> para (Pcdata),
     Para2 -> para(PCDATA), PCDATA -> pcdata end }
```

`pcdata` may be any textual value in a document that instantiates this schema.

Any Regular Tree Grammar can be expressed in the Relax NG XML schema language, which is standardized by OASIS. The key features of RELAX NG are: it is far more simple and easier to learn than XML schema, it uses XML syntax unlike DTDs and does not change the information set of an XML document (such as XML Schema). It supports XML namespaces, treats attributes uniformly with elements so far as possible, has unrestricted support for unordered and mixed content, has a solid theoretical basis, and can partner with a separate datatyping language (such W3C XML Schema Datatypes).

Example 3.2.3 (Relax NG schema) *The following schema defines some components of an Addressbook:*

```
<element name="addressBook"
xmlns="http://relaxng.org/ns/structure/1.0">
  <zeroOrMore>
    <element name="card">
      <element name="name">
        <text/>
      </element>
      <element name="email">
        <text/>
      </element>
      <optional>
        <element name="note">
           <text/>
        </element>
      </optional>
    </element>
  </zeroOrMore>
</element>
```

This may be translated into the following production rules P:

```
P = {
    Doc -> AddressBook,
    AddressBook -> addressBook (Card*),
    Card -> card (Name, Email, Note?),
    Note -> note (Pcdata),
    Email -> email (Pcdata),
    Name -> name (Pcdata),
    Pcdata -> pcdata end
    }
```

3.2.1 Local tree grammars - DTD

A restricted class of tree grammars called *local* prohibits the competition of non-terminals. This class corresponds to Document Type Definitions (DTD).

Definition 3.2.4 (Competing Non-Terminals) *Two different non-terminals A and B with $A, B \in N$ are said to be competing with each other if*

- *one production rule has A in the left-hand side*
- *another production rule has B in the right-hand side, and*
- *these two production rules share the same terminal in the right-hand side*

Definition 3.2.5 (Local Tree Grammar (LTG)) *A local tree grammar (LTG) is a regular tree grammar without competing non-terminals.*

The reader may observe that local tree grammars and extended context-free grammars (ECFG) look similar. However, the former describes sets of trees while the latter describes sets of strings. Each parse tree of a ECFG is a local tree language. Hence, our approach could easily be extended to obtain ontologies from ECFG parse trees.

In DTDs, the competition of non-terminals is prevented by not distinguishing between terminals and non-terminals. Element types are terminals as well as non-terminals and element type declarations are production rules. Content models are required to be deterministic. Attribute-list declarations associate attributes to terminals.

Example 3.2.6 (DTD) *The following DTD describes that a document my contain several paragraphs:*

```
<!ELEMENT doc (para*) >
<!ELEMENT para (#PCDATA) >
```

It can be captured by the local tree grammar G_{DTD}:

```
N = {Doc, Para, PCData}
T = {doc, para, pcdata}
S = {Doc}
P = {Doc -> doc (Para*), Para -> para (Pcdata), Pcdata -> pcdata end }
```

3.2.2 Single-Type Tree Grammars - XML Schema

A less restricted class of tree grammars called *single-type* prohibits only the competition of non-terminals within a single content model. This roughly corresponds to XML schema.

Definition 3.2.7 (Single-Type Tree Grammar) *A single-type tree grammar is a regular tree trammar such that*

- *for each $p \in P$, non-terminals in its content model do not compete with each other*
- *start symbols do not compete with each other*

The grammar G_1 established in example 3.2.2 is not single-type, since the non-terminals `Para1` and `Para2` compete and they occur in the content model of the production rule `Doc`.

Although XML-Schema has many complicated mechanisms, it is not very expressive from the perspective of formal language theory. The main features of XML-schema are complex type definitions, anonymous type definitions, group definitions, subtyping by extension and restriction, substitution groups, abstract type definitions and integrity constraints such as key, unique and keyref (key reference) constraints. The subsequent paragraphs illustrate how those features relate to our formal framework.

Example 3.2.8 (Complex type definitions) *A complex type definition defines a production rule without terminals.*

```
<xsd:complexType name="Book">
    <xsd:sequence>
        <xsd:element name="title" type="xsd:string"
                     minOccurs="1" maxOccurs="1" />
        <xsd:element name="author" type="xsd:string"
                     minOccurs="1" maxOccurs="unbounded" />
        <xsd:element name="publisher" type="xsd:string"
                     minOccurs="0" maxOccurs="1" />
    </xsd:sequence>
</xsd:complexType>
```

This example may be converted into the following set of production rules:

```
P = { Book -> (Title, Author+, Publisher?),
      Title -> title (Pcdata),
      Author -> author (Pcdata),
      Publisher -> publisher (Pcdata),
      Pcdata -> pcdata end
    }
```

Example 3.2.9 (Group Definitions) *A group definition defines a non-terminal and a production rule without a terminal.*

```
<xsd:group name="shipAndBill">
    <xsd:sequence>
        <xsd:element name="shipTo" type="Address" />
        <xsd:element name="author" type="Address" />
    </xsd:sequence>
</xsd:group>
```

This example may be converted into the following set of production rules:

```
P = { ShipTo -> shipTo (Address),
      BillTo -> billTo (Address),
      shipAndBill -> (ShipTo, BillTo)
    }
```

Example 3.2.10 (Sub-Typing) *XML schema distinguishes two forms of sub-typing, viz. extension and restriction. This leads to derived types and allows for reuse of type specifications. In the following definitions the type* `UKAddress` *is derived from* `Address` *by means of extension:*

```
<xsd:complexType name="Address">
    <xsd:sequence>
        <xsd:element name="name" type="xsd:String" />
        <xsd:element name="street" type="xsd:String" />
        <xsd:element name="city" type="xsd:String" />
```

```
    </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="UKAddress">
    <xsd:complexContent>
        <xsd:extension base="Address">
            <xsd:sequence>
                <xsd:element name="postcode" type="xsd:string" />
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
```

The definitions of Address and UKAddress can easily be translated into two production rules `Address -> (Name, Street, City)` and `Address -> (Name, Street, City, Postcode)` To reflect the specified type subsumption, all subtypes must (transitively) be instantiated whenever a supertype is referenced in a production rule. An element declaration `<xsd:element name="shipTo" type="Address">` must therefore be translated to two production rules `ShipTo -> shipTo (Address)` and `ShipTo -> shipTo (UKAddress)`

Example 3.2.11 (Substitution Groups) *Substitution groups establish equivalence classes and specify which elements may take the place that has initially been specified by another type:*

```
<element name="shipComment" type="Y" substitutionGroup="comment" />
<element name="customerComment type="Z" substitutionGroup="comment" />
```

The resulting production rules are:

```
P = { ShipComment -> shipComment Y,
      CustomerComment -> customerComment Z,
      comment -> shipComment Y,
      comment -> customerComment Z    }
```

Integrity constraints The means of XML-Schema for capturing integrity constraints such as key, unique and keyref (key reference) cannot be captured by tree grammars and remain outside of our translation approach for the time being. Interestingly, no formalization of those concepts have yet been given. XML Schema itself, being a large and complex standard of more than 300 pages in printed form, remains only partially formalized (cf. [6], [27] for those partial attempts). The largest problem with respect to key references is that the links between XML elements are not typed, hence only a generic link to any other class could be created.

3.3 Lifting Process

The lifting process tries to capture the semantics of the XML schema by translating non-terminals and terminals to concepts and roles in the ontology. The translation process sequentially processes each rule.

Domains and Ranges The left-hand side of each rule is the domain of the role. The content-model is the range of the role. In case of complex content models, where more than one non-terminal is involved in the regular expression the rule is expanded one step via a derivation relation. For example

```
P = { ShipTo -> shipTo (Address),
      BillTo -> billTo (Address),
      shipAndBill -> (ShipTo, BillTo)
    }
```

is expanded to

```
P' = {shipAndBill -> (shipTo (Address)), billTo (Address))}
```

In this case the left-hand of the expanded rule is the domain of all roles in the expanded rule.

Attributes Attribute definitions are treated like terminals and converted to appropriate production rules, e.g.

```
<!ELEMENT square EMPTY>
<!ATTLIST square width CDATA "0">
```

is translated to the following production rules

```
P = {
    Square -> square end
    Square -> width (CDATA)
    CDATA -> cdata end
    }
```

Rules with the terminal symbol *end* are not translated. PCDATA and CData are translated to rdf:Literal. In future versions of OntoLift the appropriate simple datatypes defined in XML schema will be translated appropriately.

Cardinalities The cardinality information from the regular expressions defining the content models are preserved. Table 1 shows how the translation to cardinality definitions in the ontology is performed.

Reg. Expr. Quantifier	Min Cardinality	Max Cardinality
	1	1
?	0	1
+	1	unlimited
	0	unlimited

Table 1: XML Cardinality translation

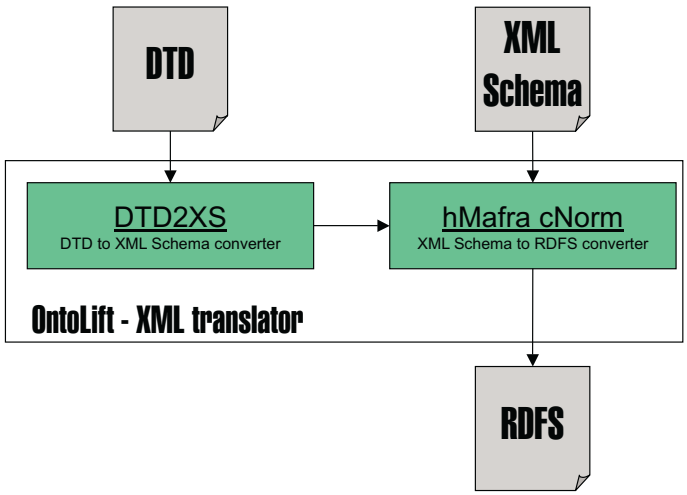

Figure 3: Architecture of the OntoLift XML component

3.4 Prototype Implementation

All terminal and non-terminal symbols are valid URIs. Namespaces are treated appropriately as prefixes to XML QNames. The implementation is currently based on the hMafra tool implemented by Oliver Fodor in the EU-funded Harmonise project[8]. The hMafra tool implements a lift for XML Schema to RDFS ontologies. We can realize the lifting of DTDs to ontologies via an intermediate tool that provides a DTD to XML Schema conversion, e.g. dtd2xs[9]. The architecture of the XML-translation component in OntoLift is depicted in Figure 3.

Limitations The prototype is currently limited to RDFS web ontologies. Hence, cardinality information is not preserved. The tool is also not aware of XML Schema Simple Datatypes. Ranges of roles where XMLS simple datatypes are involved are assigned rdf:Literal as a range.

3.5 Further Applications

The OntoLift tools may be applied to enhance the semantic description provided in WSDL web service interface descriptions. In WSDL parameters and return values of the methods provided by a web service are typed by means of XML schema. The automatic translation

[8] available for download at http://sourceforge.net/projects/hmafra/
[9] available for download at http://puvogel.informatik.med.uni-giessen.de/lumrix/

of those types to concepts in an ontology allow for further refinement of the semantics of such a datatype and may be used for improving the search for services based on the formal and semantic descriptions of the input and output values delivered by a service.

3.6 Related Work

Within the EU-funded project Harmonise a reconciliation tool has been implemented, which aims to reach interoperability between existing standards for exchange of tourism data. This is achieved by lifting existing XML schemata to initial ontologies and providing mappings between those ontologies [15]. Another approach for mapping XML schemas to DAML ontologies is provided by [23].

Both approaches build on XML Schema only and provide no generic means for translating arbitrary schemata based on regular tree grammars.

4 Object-Oriented Specifications

This section presents the transformation of UML diagrams to ontologies. The Unified Modeling Language (UML) has been widely adopted by the software engineering community and its scope is broadening to include more diverse modeling tasks. We can build on the results of several approaches which argue that UML is a well-suited notation for the development of ontologies.

4.1 Prerequisites

UML was originally designed for human-to-human communication of models for building systems in object-oriented programming languages. UML is an open standard proposed by the Object Management Group (OMG), that also works on a Model Driven Architecture (MDA) based on UML and related standards such as the Meta-Object Facility (MOF) and XML Metadata Interchange (XMI). This MDA is driving UML to become more formal and machine-processable so that models can be used at compile time and runtime and not just as a graphical notation for human-to-human communication. While UML itself is given no formal semantics, several researches have been working into this direction[10]. The lack of semantics makes the lifting process less formal than the previous approaches.

4.1.1 Limitations

UML is a large standard and only small parts may be considered as a specification of ontological information. We consider only class diagrams and Object Constraint Language (OCL) constraints as relevant input for the transformation process. Other UML diagrams such as state charts and Activity Diagrams are more process-related and therefore not considered here.

[10] see for example the proceedings of an OOPLSA workshop dedicated to that purpose [21] and the RFI submission for UML 2.0 by the precise UML group [9]

Even if we consider only UML class diagrams as input for the mapping process several of the available features remain outside of the transformation, i.e. method and scope specifications as well as association classes.

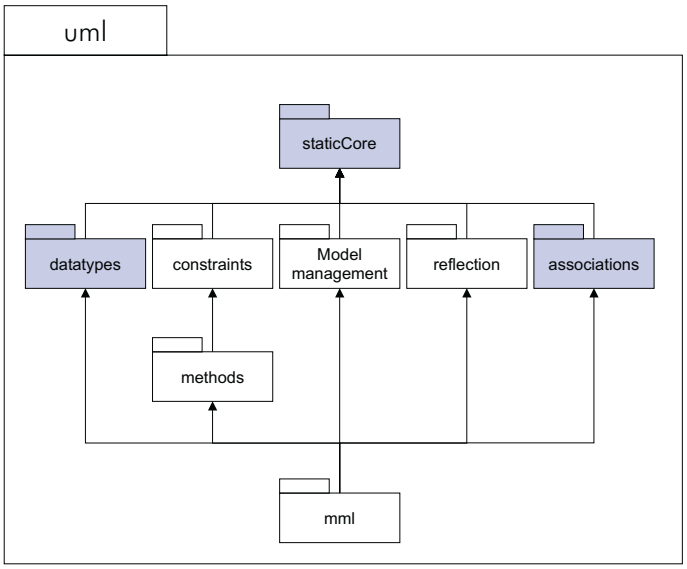

Figure 4: Components of UML in MML

4.2 MML - A precise UML variant

Due to the unspecified formal semantics of UML, we base our mapping on the formalization of UML provided by the Meta-Modelling Language MML [10].

We only give a brief introduction into MML, the interested reader may refer to [10] for a comprehensive introduction. The description of UML given by MML is component-based and uses the established graphical UML notation to define constructs (such as done in UML/MOF itself). However, all constructs are amended with well-formedness rules (stated in a restricted, formalized OCL variant). Figure 4 shows which components are relevant for our endeavor (marked in blue/shaded in grey).

Static core The static core package describes the fundamental static modelling constructs required to build UML models. It provides a general framework for extending MML with new model elements, presenting several plug-in points for extensions. Hence not all features must be mapped.

The general modelling concepts are presented in Figure 5. We are only interested in two aspects: class and attribute. We observe that classes may have multiple attributes and that attributes are also considered classifiers in UML since they are viewed as containing their types. Both classes and attributes must have names since they are ModelElements. This disallows anonymous classes and attributes, which are allowed in some UML modelling environments. Attributes are not generalisable, since additional OCL constraints (not shown here) forbid that. However for classes generalisation is allowed and we can

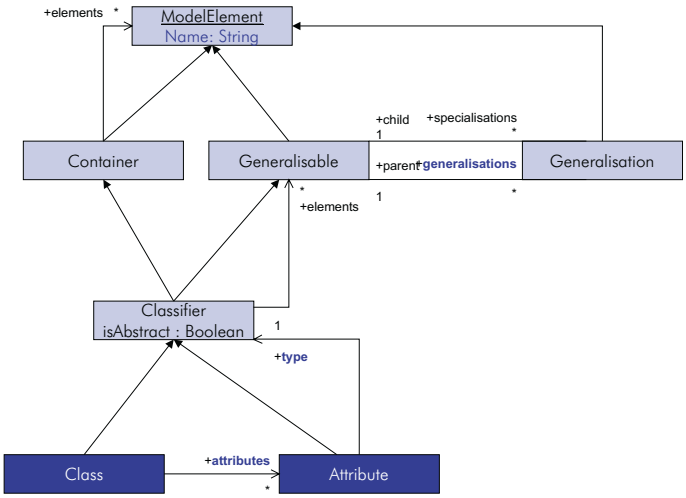

Figure 5: MML UML Core Modelling concepts

interpret the generalisations association depicted in the Figure as a concept hierarchy in the ontology.

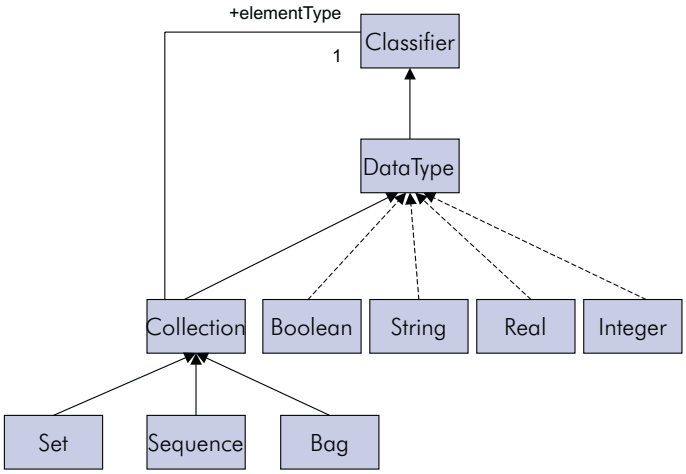

Figure 6: MML - UML datatypes

Data types The data type package of MML builds on the static core model package. Figure 6 presents the MML modelling of UML data types. Data types are also considered as classifiers like most other UML constructs. The package brings in some built-in types (such as integer and string) as well as a collection datatype to represent the UML collection types. Usually UML allows for multi-valued attributes. The MML modelling variant models multi-valued attributes by making the type of an attribute a collection type. Each collection is parameterized with the type they contain and additional OCL constraints ensure that the elements of a collection have the correct element type. In the ontology Collections may be translated to a data type role whose range is the data type that is specified for the collection. However we will not be able to distinguish between Sets and Bags

uml	daml
class	class
data type	xml schema datatype
Collection	xml schema datatype specified as elementType
association	object property
assoc. multiplicity	cardinality restrictions
role	object property
attributes	datatype property
generalization	subClassOf
import	import
package hierarchy	namespace

Table 2: UML to ontology translation

and Sequences. The translation of built-in data types to the corresponding XML data type is straight forward.

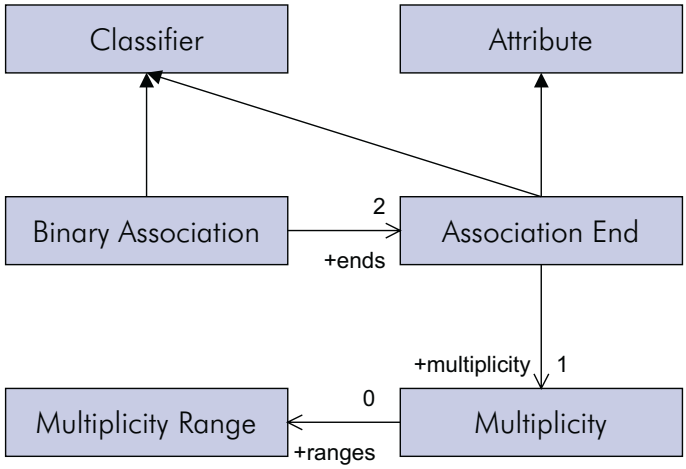

Figure 7: MML - UML associations

Associations The associations component in MML is only partially relevant for our purpose, since the UML notion of association classes has no resemblance in ontologies. Association classes specify additional attributes about the association itself. This cannot be represented in Web ontologies. Figure 7 presents the MML modelling of those UML associations, which may be mapped.

Binary UML associations may easily be transformed to roles in ontologies, since they model bi-directional links between classes. In MML a binary association is represented represented by a pair of association ends, which are essentially attributes extended with multiplicity information. This multiplicity information may directly be translated to cardinality restrictions in Web ontology languages

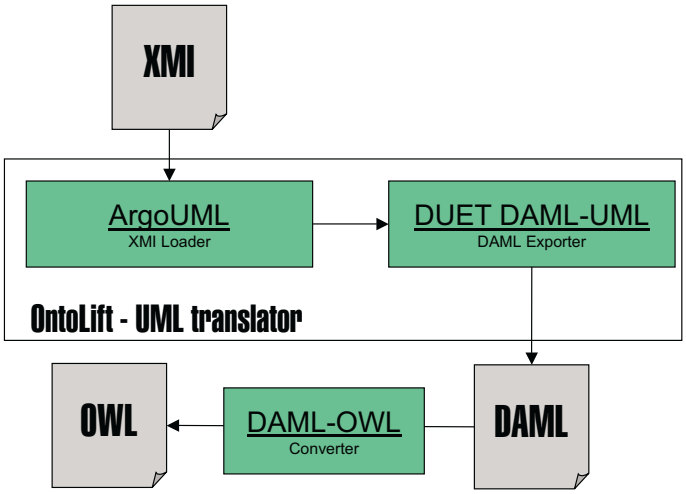

Figure 8: Architecture of the OntoLift UML component

4.3 Translation rules

Since we have already discussed the translation from UML to ontologies wrt. to the modelling constructs, we only need to summarize our findings. This is done in table 2. We take some additional assumptions in the translation process: Package hierarchies are translated to XML namespaces, all names are again prefixed with a user specified URI prefix. Imports in XMI files are translated to appropriate imports in OWL Web ontologies.

4.4 Prototype Implementation

The OntoLift prototype of the UML translation is in a very early stage. It builds on the DUET UML-DAML mapping tool provided by the DARPA-funded DAML initiative. DUET provides a plug-in for visual UML editors such as the commercial Rational Rose or the open-source ArgoUML editor and allows to export given UML files into DAML files.

The architecture of prototype is depicted in Figure 8. Our translator reads XMI files by means of the ArgoUML XMI libraries. The XML Metadata Interchange Format (XMI) allows a Stream-based Interchange of UML and MOF models between applications. The loaded UML model is then exported to DAML via the DUET tool. The DAML ontologies are then converted to the OWL syntax.

4.5 Further Applications

The UML translation can be used as a basis for developing a visually intuitive Web ontology environment.

4.6 Related Work

A variety of different research projects and commercial initiatives have been applying UML for ontology representation. [13] have investigated the use of UML class diagrams

for representing ontologies and UML object diagrams for representing instance knowledge. [24] have proposed an approach to agent-oriented software engineering based on the use of UML to model various aspects of a multi-agent system. One of their proposed diagrams is an ”ontology diagram”, which depicts classes representing agents and domain entity types, and associations representing domain predicates that can be encoded as KIF or FIPA-SL agent message content. The UML based Ontology Tool-set (UBOT) project is building ontology engineering and natural language processing-based text annotation tools for DAML. UML is used as a front-end for visualizing and editing DAML ontologies. The approach is to extend UML by defining a prototype UML profile for DAML which maps UML stereotypes to DAML-specific elements [1]. The DAML-UML Enhanced Tool (DUET), which is based on available UML editors, adds a UML profile for DAML [1]. Sandpiper Software is developing commercial grade tools that support knowledge modelling and information brokering [16]. Sandpiper has extended UML to enable representation of rich ontological knowledge through the creation of a UML profile for frame-based knowledge representation.

5 Conclusion and Outlook

We have presented translations from popular legacy schema formats such as relational database schema, XML schemas and UML software specifications. The prototype provides first implementations of converters. In the future we will add further translators from thesaurus standards to leverage the large resources put in place by many publishing companies and (public) libraries.

Another direction will be the usage of simple heuristics to align the ontologies to the top-level foundational ontologies developed in work package 3. In the long run we envision a generic schema registry where people can upload their models and that offers a set of services such as search, subscription and a set of available translators between standards such as developed in this deliverable.

References

[1] Baclawski, Kokar, Kogut, Hart, Smith, Holmes, Letkowski, and Aronson, *Extending umlto support ontology engineering for the semantic web.*, Proc. of 4th Int. Conf. on UML - UML2001, October 2001.

[2] C. Batini, S. Ceri, and S. Navathe, *Conceptual database design - an entity-relationship apporach*, Benjamin/Cummings Publishing Company, 1992.

[3] A. Behm, A. Geppert, and K. Dittrich, *On the migration of relational schemas and data to object-oriented database systems*, Proceedings of 6th Int. Conf. on Extending Database Technology (EDBT), Valencia, March 1998, pp. 436–450.

[4] J. Biskup, *Achievements of relational database schema design theory revisited*, Semantics in Databases (B. Thalheim and L. Libkin, eds.), LCNS, Springer, 1998, pp. 29–54.

[5] T. Bray, J. Paoli, and C. Sperberg-McQueen, *Extensible markup language (xml) 1.0*, Recommendation, W3C, October 2000, http://www.w3.org/TR/REC-xml.

[6] A. Brown, M. Fuchs, J. Robie, and P. Wadler, *Msl - a model for xml schema*, WWW 10, Hong Kong, China, May 2001.

[7] R. Chiang, T. Barron, and V. Storey, *Reverse engineering of relational databases: Extraction of an eer model from a relational database*, Journal of Data and Knowledge Engineering, vol. 12, Elsevier, March 1994, pp. 107–142.

[8] J. Clark, J. Cowan, M. Fitzgerald, K. KAWAGUCHI, J. Lubell, M. MURATA, N. Walsh, and D. Webber, *Document schema definition languages (dsdl) part 2: Grammar-based validation relax ng*, Enquiry ISO/IEC DIS 19757-2, ISO/IEC.

[9] T. Clark, A. Evans, R. France, S. Kent, and B. Rumpe, *Response to uml 2.0 request for information*, Tech. report, The Precise UML Group, December 1999, http://www.cs.york.ac.uk/puml/papers/RFIResponse.PDF.

[10] Tony Clark, Andy Evans, Stuart Kent, Steve Brodsky, and Steve Cook, *A feasability study in rearchitecting uml as a family of languages using a precise oo metamodelling approach*, Tech. report, IBM Corporation and The Precise UML group, Internet: http://www.cs.york.ac.uk/puml/mmf/mmf.pdf, September 2000.

[11] S. Comai and P. Fraternali, *A semantic model for specifying data-intensive web applications using webml*, International Semantic Web Working Symposium (SWWS), 2001.

[12] H. Comon, M. Dauchet, R. Gilleron, F. Jacquemard, D. Lugiez, S. Tison, and M. Tommasi, *Tree automata techniques and applications*, Internet: http://www.grappa.univ-lille3.fr/tata, 1997.

[13] S. Cranefield and M. Purvis, *Uml as an ontology modeling language*, Workshop on Intelligent Information Integration, 16th Int. Joint Conference on AI (IJCAI-99), 1999.

[14] C. Date, *Referential integrity*, Proceedings of Intl. Conf. on Very Large Data Bases (VLDB), 1981, pp. 2–12.

[15] M. DellErba, O. Fodor, F. Ricci, and H. Werthner, *Harmonise: A solution for data interoperability*, I3E 2002, 2002, pp. 433–445.

[16] M. Dutra, *Uml for knowledge representation*, 2nd Workshop on UML for Enterprise Applications, December 2001.

[17] J.-L. Hainaut, *Database reverse engineering: Models, techniques and strategies*, 10th Conf. on ER Approach, San Mateo (CA), 1991.

[18] S. Handschuh, S. Staab, and A. Mdche, *Cream - creating relational metadata with a component-based, ontology-driven annotation framework*, Proceedings of K-CAP 2001, October 2001.

[19] V. Kashyap, *Design and creation of ontologies for environmental information retrieval*, 12th Workshop on Knowledge Acquisition, Modeling and Management (KAW'99), October 1999.

[20] N. Klarlund, A. Moller, and M. Schwatzbach, *Dsd: A schema language for xml*, ACM SIGSOFT Workshop on Formal Methods in Software Practice, August 2000.

[21] L.Andrade, A. Moreira, A. Deshpande, and S. Kent (eds.), *Oopsla'98 workshop on formalizing uml. why? how?*, OOPSLA 1998, 1998.

[22] M. Murata, D. Lee, and M. Mani, *Taxonomy of xml schema languages using formal language theory*, Extreme Markup Languages 2000, 2000.

[23] Mark Neighbors, *Xml to daml translator*, Presentation at DAML PI meeting, July 2001, Available online at: http://www.davincinetbook.com:8080/daml/xmltodaml/xmltodaml.html.

[24] F. Bergentiand A. Poggi, *Exploiting uml in the design of multi-agent systems*, (2000).

[25] N. Rishe, *Database design: The semantic modelling approach*, McGraw-Hill, 1992.

[26] S. Abiteboul S., R. Hull, and V. Vianu, *Foundation of databases*, Addison-Wesley Publishing Company, 1995.

[27] J. Simeon and P. Wadler, *The essence of xml*, POPL'03, New Orleans (LA), USA, January 2003.

[28] Ljiljana Stojanovic, Nenad Stojanovic, and Raphael Volz, *Migrating data-intensive web sites into the semantic web*, Proceedings of the 17th ACM symposium on applied computing (SAC), ACM Press, 2002, pp. 1100–1107.

[29] M. Takahashi, *Generalizations of regular sets and their application to a study of context-free languages*, Information and Control, vol. 27, January 1975, pp. 1–36.

[30] H. Thompson, D. Beech, M. Maloney, and N. Mendelsohn, *Xml schema part 1: Structures*, Recommendation, W3C, May 2001, http://www.w3.org/TR/xmlschema-1/.

[31] J. van Griethuysen, *Information processing systems – concepts and terminology for the conceptual schema and the information base*, Tech. Report ISO/TR 9007:1987, ISO JTC 1/SC 32, International Standards Organization (ISO), 1987.

[32] M. Vermeer and P. Apers, *Object-oriented views of relational databases incorporating behaviour*, Prc. of the 4th Int. Conf. on Database Systems for Advanced Applications (DASFAA), April 1995, pp. 26–35.